Message Passing in Mobile Environments

Priya Venkitakrishnan, Sharan Raman, Smita Hegde &
Sreekanth Narasimhan
Department Of Computer Science and Engineering, University Of Texas At Arlington.
Project Paper for Advanced Operating Systems Course, Spring 2002.

Abstract

The widespread development of distributed systems enable geographically dispersed teams to work together, supporting communication, coordination and collaboration. This paper presents a report on how such an environment is developed using the message-passing paradigm. The working set up for the collaborative environment is designed in a Windows CE environment to enable group coordination among users with mobile PCs or iPAQs. In this paper we propose Peer-to-Peer Architecture with Centralized Control for this application. We also describe the procedures for the operation of the framework under various scenarios. Finally we analyze the proposed architecture describing the important features that it supports.

1 Introduction

The emergence of the World Wide Web has provided different means for coordination and collaboration among geographically dispersed teams. The current trend is to support such group collaboration and communication in a wireless environment that permits individuals who are not just connected to the network but also to others who are on the move. The two main paradigms used for this group collaboration are Message Passing and Shared Memory. This paper talks about the Message Passing paradigm and explains how Message Passing can be implemented in a mobile environment to support group coordination and communication.

Section 2 of the paper introduces the Windows CE environment and the embedded tools used to develop a framework for the collaborative environment. Section 3 provides a literature review of the Message Passing paradigm and brings out its fundamental concepts. Section 4 gives a detailed description of our project design and brings out how the design takes care of all issues related to collaboration in a mobile environment. Section 5 talks about the main features supported by the proposed architecture and an analysis on the design.

2 Windows CE Operating System And Visual Development Tools

In the last few years computing devices have emerged in the commercial market that provide unparalleled portability and ease of accessibility. The desktop PC capabilities have been cut down to a really small footprint. Various mobile devices such as Pocket PC, Palm Pilot and PDA are creating an impact in the software industry. Microsoft Pocket PC has tremendous features for enterprise computing, games, and entertainment. The Windows CE operating system has been embedded into many appliances and custom devices for monitoring and control. Desktop Windows CE devices are available that provide thin-client computing. They have Windows Terminal Server client installed, allowing them to effectively run Windows NT and 2000 applications. Being thin clients, they are easy to set up and maintain.

2.1 Windows CE – The Embedded Operating System

Windows CE is a modular operating system based on Microsoft Windows, designed particularly for embedded applications. Windows CE is a 32 bit preemptive Multitasking operating system offering fully protected memory spaces [12]. It provides a Graphical User Interface similar to that of the common desktop Windows environment, but eliminates a lot of features and controls from its close relative. The Win32 API of the WinCE OS is smaller than the desktop Windows API. It provides Touch Screen APIs that allow users to interact with the touch sensitive display present on handheld devices [12].

Some of the Windows CE operating system characteristics and capabilities are: [10]

- Windows CE SDKs, such as Pocket PC 2002, support an emulation environment on the PC.
- Multiprocessing, multithreaded support with synchronization.
- Virtual memory architecture.
- File system and property database support, database access through ADOCE (ActiveX Data Objects for Windows CE)
- TCP/IP stack with functions allowing HTTP and socket communication, access to Windows NT and 2000 network resources
- Serial port communications
- COM (Component Object Model) and DCOM (Distributed Component Object Model) support
- Synchronization of data with desktop PCs using ActiveSync
- Development of applications in eMbedded VB, eMbedded VC++ using Microsoft Foundation Classes and Pocket PC SDK.

Salient features such as portability, small software footprint and ease of use has propelled the use of Windows CE in a wide gamut of devices like handheld computers, cable TV boxes, Auto PCs and Mobile Phones [13]. Some of the famous Handheld PCs that use Windows CE are Compaq's iPAQ, HP's Jornada, Sony's Clie and Casio's Cassiopeia [11].

2.2 eMbedded Visual C++ and eMbedded Visual Basic

In the past, Microsoft offered add-ins for Visual C++ to provide a Windows CE development environment. The main problem with the add-ins approach was that the Windows CE applications included all the facilities that came with Windows NT/98/2000 applications that were not really relevant for the Handheld.

eMbedded Visual C++ 3.0 (eVC++) is a new programming tool specially designed to write Windows CE applications for any target device for which there exists a Software Development Kit (SDK). The Microsoft Platform Builder can also produce an SDK for that device and this can then be installed in eVC++ and applications can be developed for that device. Most of eVC++ is based on VC++ and shares the same user interface, but only the tools and facilities necessary for writing Windows CE applications are present.

We have used eMbedded VB for our application. Microsoft's eMbedded Visual Basic (eVB) looks very much like the full Visual Basic, but it is a based on VB Script. The editor supports code completion and Option Explicit. Setting a reference to a type library enables the object browser for the specified library, as well as code completion for the methods and properties presented by its objects. File handling functions are supported in eVB and you can call the Windows CE API or any DLL procedure by using the Declare statement [8]. It is slower than eVC++, but the deficiencies can be minimized by use of native code controls and DLLs [8]. There are a number of intrinsic controls including the GUI essentials and ActiveX controls that are available out-of-the-box. There are no bound controls, but ADOCE is available for data access.

2.3 Emulation Environments

Many Windows CE SDKs, such as Pocket PC, support an emulation environment that runs on the desktop PC. The Windows CE application is generally built using Embedded Visual tools on the desktop and then is tested using the Emulator. Once the application is completely built it can be transferred to the Mobile device. The results of an emulator are not always perfect. Facilities such as networking, RAS dialup connections and user interfaces may behave differently on the emulator. But using emulation does save a lot of development time while debugging non-user-interface code. It is quicker and easier to use than downloading applications onto a real device.

2.4 Microsoft ActiveSync

Microsoft ActiveSync is the latest synchronization software for Microsoft Windows CE powered Pocket PCs [14]. It allows users to create a partnership between the mobile device and the desktop computer using an USB cable or infrared port [15]. The creation of partnerships synchronizes information between the two

devices keeping the information up to date. Any change of information on one device is immediately reflected on the other. It is used to transfer and synchronize files and data and also used to upload application executables onto the Mobile device.

3 Message Passing Paradigm – A Literature Review

3.1 Introduction

"A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages " [1]. In a distributed system, computers geographically distributed over a network collaborate and communicate to achieve a particular task. Different models of computer architecture that have been designed based on the communication paradigm are:

- **Message Passing Model:** In this model, information is conveyed from a specific sender to one or more receivers. This is analogous to making phone calls or sending letters [2]. The Message Passing model is explained in detail in the following sections. This is totally a shared nothing model.
- **Shared Memory Model:** Here the nodes share a global address space and communicate by writing to the global space [4]. A typical representative of this model is a bulletin board where groups of people can share their ideas by posting information at a common shared location [2].
- **Remote Service Model:** In this model, requests for accesses/services are delivered to the server machine, which executes the request and delivers the results to the user [3]. The remote service is provided using a message based communication scheme [3]. A good example of this model is Java RMI.
- **Threads:** A single program of execution can be divided into multiple threads each sharing a common address space. In this model of interprocess communication, a thread can be asked to send and receive messages while other threads of the same task can continue with their operations [3].
- **Data Parallel Processing Model:** In this model, different server agents process subset of a given dataset simultaneously and exchange the final result [2]. The exchange of information can be implemented using a message passing or a shared space model.

3.2 Message Passing Model

In this model, each processor has its own private or local memory [4]. There is no concept of a shared memory. The processors need to communicate via a communication network. In this model, when a processor needs to access data present in another processor, an explicit request must be made by the processor to transfer the data through the communication network.

The main function of a message passing system is to allow processes to communicate with each other. The most common user level primitives used by message passing systems include the SEND and RECEIVE operations [1][2][3].

When two processes P and Q need to communicate, they send and receive messages to and from each other. For this, a communication link must exist between the two processes. There are several ways of establishing links and implementing the send and receive operations [3]. This is discussed below.

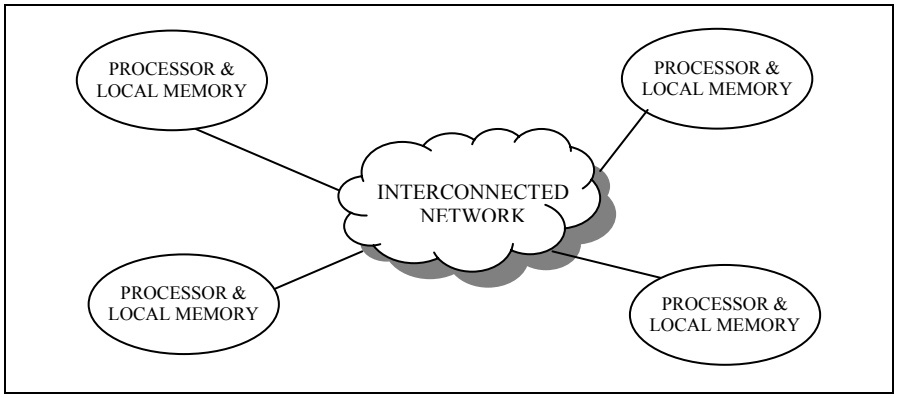

Figure 1: Message Passing

3.2.1 Direct Communication

A link is directly established between every pair of processes that needs to communicate. The send and receive operations may be represented as,

Send (destination-process, message)

Receive (source-process, message)

Both the sender and the receiver must know the identities (names) of the sending and the receiving process. The sender appends its messages to a remote queue at the destination and the receiving processes read the messages from their local queues [1]. The communication in this mode can be carried out in two ways : *Synchronous (Symmetric)* communication and *Asynchronous (Asymmetric)* communication [1] [3].

In the *Synchronous* form of communication, both the sending and receiving process need to synchronize at every step while communicating with each other [1]. Here, the sender is blocked after issuing a send operation until the receiving process receives the message and sends back an acknowledgement. The Synchronous communication is more time consuming but is easier to implement.

In the *Asynchronous* form of communication, sending operation is non-blocking, i.e., the sender can continue with its normal operations after it issues a send and the message is copied to a local buffer [1]. The receive operation can be either blocking or non-blocking. In the blocking receive, the receiver must suspend its normal execution and wait till it receives the entire message. In the non-blocking receive, the receiver can proceed with normal execution and the arriving messages are queued to an input buffer [1]. Asynchronous communication is more efficient, but it is much harder to implement and may pose problems related to concurrency.

3.2.2 Indirect Communication

The communication is carried out using objects called the “Mailboxes” to which messages are sent to and received from [3]. For two processes P and Q to communicate they must be associated to a common mailbox. The send and receive primitives mat be defined as:

Send (mailbox X, message)

Receive (mailbox X, message)

Here, both the send and receive primitives need to only know the mailbox address and need not know the identities of the processes involved. In the above-mentioned primitives, a message is sent to mailbox X and stored there. A mailbox maybe owned by a single process or maybe owned by multiple processes. The former provides a one to one communication whereas the latter provides one to many communications.

3.3 Interprocess Communication in Distributed Systems

The process interaction pattern in distributed systems may be described using three basic paradigms as follows [6]:

1. **Producer/Consumer model**: In the producer/consumer model we have a group of processes called the Producers, which periodically produce items consumed by the consumers.
2. **Client/Server Model**: The client/server form of communication is done in a request reply manner in which the client process is normally blocked until it receives the reply from the server. This is mostly a reliable form of communication in which the reply from the server is like an acknowledgement to the client [1].
3. **Interacting Peers**: This model has a group of people such as managers/workers who communicate and coordinate to achieve a specific task. A detailed description of our problem statement and the chosen design is explained in section 4.

3.4 Group Communication [1]

The invention of IP based networking provides end users to communicate over a distributed network by sending mails, browsing the World Wide Web etc [7]. For group communication a pair wise exchange of messages is not recommended due to the overhead on the sender and the network [1] [7]. The group communication is implemented using a Multicast operation. "Multicasting enables a group of hosts to communicate and share the information" [7]. There are different variations of Multicasting used, based on the application requirements [1]. The simplest one provides no guarantees and is unreliable [1].

Multicasting may be efficiently used to multicast client request to different servers, notify the occurrence of an event to a group of users, to discover services in spontaneous networking etc [1]. For group communication, a type of multicast called the IP Multicast is most commonly used. The following section will briefly explain the IP Multicast. Other types of multicast exist such as the Basic Multicast, Reliable Multicast, and Ordered Multicast, which are beyond the scope of this report.

3.4.1 IP Multicast

In IP Multicast, the sender can transmit a copy of its message through the IP network to all members of the group [7]. For IP Multicast, the sender need not be aware of the identities of the individual recipients of the group [1]. Any packet multicast is sent to all members of the group. Members can join or leave the group any time they wish [1]. The basic components associated with IP Multicast model are:

Internet Group Management Protocol (IGMP): A group membership protocol that manages the multicast group [7]. The requests from hosts to join or leave the group is managed the IGMP.

Routers: The internet routers uses the Distance Vector Multicast routing Protocol (DVMRP) or the Protocol Independent Multicast (PIM) to build a group specific delivery tree connecting the sender to all the members of the group [7].

The IP Multicast is provided via UDP or the Unreliable Datagram Protocol [1]. Hence the main disadvantage faced by IP Multicast is that it is unreliable and provides no guarantees on packet delivery to all members of the group. [7]

4 Project Description and Design

4.1 Project Description

The objective of the project is to develop a platform for mobile devices to communicate and collaborate in a Windows CE environment. To achieve this, we consider a scenario where groups of people in a corporate environment using Compaq iPAQs need to schedule a meeting. We need to develop a framework that is based on the Message passing paradigm to schedule meetings. The group might include a Project Manager at the highest level and others at a subordinate level. The members of the group are mobile and hence require wireless communication. A unique characteristic of this application is that the members are always on the move and hence are often disconnected from the wired network. They may abruptly go out of range while being hooked to the network. We intend to develop a Windows CE application to simulate the above-mentioned working environment.

4.2 Peer to Peer Architecture with a Centralized Control

4.2.1 Architecture Overview

We propose an architecture based on the message passing paradigm to allow mobile users to coordinate meetings. We use a *Peer-to-Peer* communication model where every mobile user can directly talk to any other mobile user in that group. This is based on a hybrid of *Direct communication and Indirect communication* version of Message passing. Refer figures 2 and 3.

As the mobile devices are not constantly connected to the network, they do not have fixed IP addresses. They obtain a new IP address every time they hook on to the Network. Since the IP addresses of mobile users change dynamically with time, the identity of devices need to maintained by a Centralized Control. As a result the system components include the Mobile users (having iPAQs) and a Centralized Control Name Server.

A broad overview of the working of the framework is as follows: The fixed Identities (assigned to every iPAQ during the formation of the group) are maintained in the Name Server. Whenever a Mobile user hooks on to the network, the application sends the Dynamic IP address to the Name server. The Name server associates this address with the fixed identity of that mobile user. It also distributes this association to all other Active mobile users. When the iPAQs need to communicate, they send messages directly to each other using the current IP address information (distributed by Name Server). This is the *Direct Message* passing scheme. However if the receiver is not active (meaning he is not hooked onto the network) the message is sent to the Name Server where it gets stored till the receiver becomes active again. This is a form of *Indirect Message* passing scheme.

The following diagrams depicts this architecture:

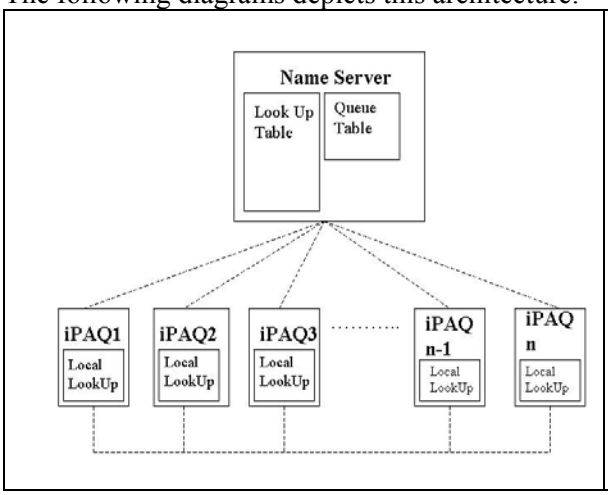

Figure 2: Connection of all the iPAQs to the Name Server

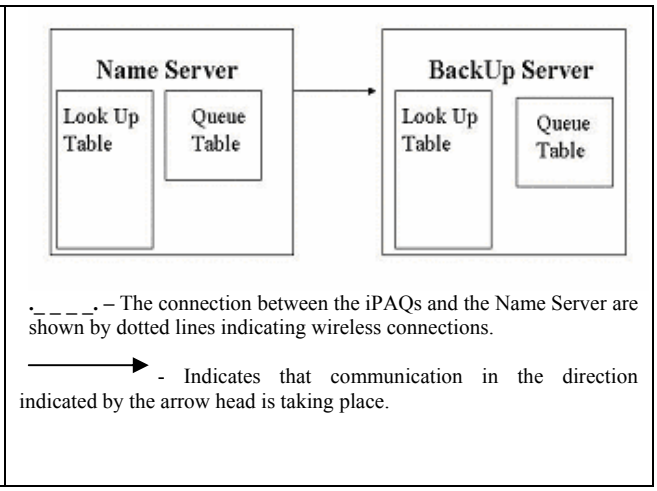

Figure 3: Back Up Server in case of Name

4.2.2 Components of the System

- **Name Server –** As seen in figure 2, this is the main server, which keeps record of how many users having Ipaqs are in the group. It maintains two tables, the Look Up Table and the Queue Table. The Look Up Table has the list of users and their respective passwords. It also has their current IP address and the current operational state. The Queue Table is like a mailbox for the user. It stores the missed messages of the user.

    - **Look Up Table** - This table will be maintained accurately and will be updated as soon as any change of state is observed. **iPAQIDnumber** is the identification of the user and the **password** stores **password** for authenticating the user. **currentIPaddr** fields stores the current IP address of the iPAQ. The default address kept is 000.000.000 indicating the state of the iPAQ as "Not active", meaning that the Ipaq is not connected to the network. The **state** field is "Active" if there is a valid IP address meaning that the iPAQ is connected to some network and can communicate. A sample of the table is given below:

Look Up Table

IPAQIDnumber	password	currentIPaddr	State
IPaq123Smita	123abc	123.123.123	Active
Ipaq999Sharan	123abc	000.000.000	Not Active
Ipaq444Priya	123abc	000.000.000	Not Active
Ipaq777Sree	123abc	122.222.222	Active

As seen in the table, there are 4 members in the group. Thus only these members can connect and join the group. Two of them are currently active while the other two are not.

- **Queue Table –** This table stores the missed messages, i.e, the messages which could not be delivered due to the Ipaq being inactive or out of range. **numOfMissedMgs** store the count of messages missed by the iPAQ due to its being in the "Not Active" state or suddenly going out of range which is an essential feature of Mobile Devices. **messageNo1** stores the first missed message.

Queue Table

iPAQIDnumber	numOfMissedMgs	MessageNo1	
IPaq123Smita	0	No Msg	
Ipaq999Sharan	2	Can we all meet at 1.00pm today	The meeting has been fixed at 2.00pm.
Ipaq444Priya	0	No Msg	
Ipaq777Sree	0	No Msg	

All the Ipaqs which are not active (these have no current IP address as they are switched off or not in range, or not connected etc) have their messages stored in the Name server. These can be retrieved when the iPAQ starts up again.

- **iPAQ -** The figure 2 shows a user with an iPAQ, who is mobile and connects to different networks. The IP address keeps changing as the mobile device changes the physical location. It stores a local copy of the Look Up table, which is modified by the Name Server whenever such an event occurs.

  A Sample Local Look Up table of iPAQ1 is shown below:

iPAQIDnumber	currentIPaddr	state
Ipaq2haran	000.000.000	NotActive
Ipaq3Priya	000.000.000	NotActive
Ipaq4Sree	122.222.222	Active

Every iPAQ will have its own local look up copy of the members and their IP addresses. This table is a replica of the table maintained by the server with a few modifications. Only the server does any kind of updation in the table. The Name Server periodically updates each IPAQ's local Look-Up Table by sending Update messages.

4.3 Operational Procedures

- **Adding and Deleting Members**

  As seen, for any user to connect and communicate with the group he or she has to be added in the Look Up Table maintained by the Name Server. This has to be manually done by the system admin. The user ID and the password are given by the System Admin.

- **Start Up Procedure**

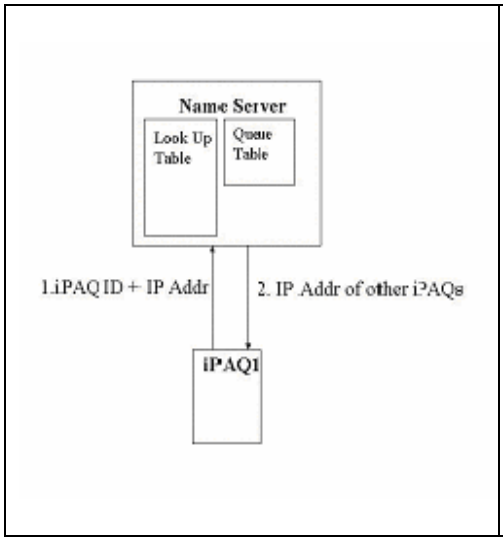

Figure 4: Start up Procedure for an iPAQ

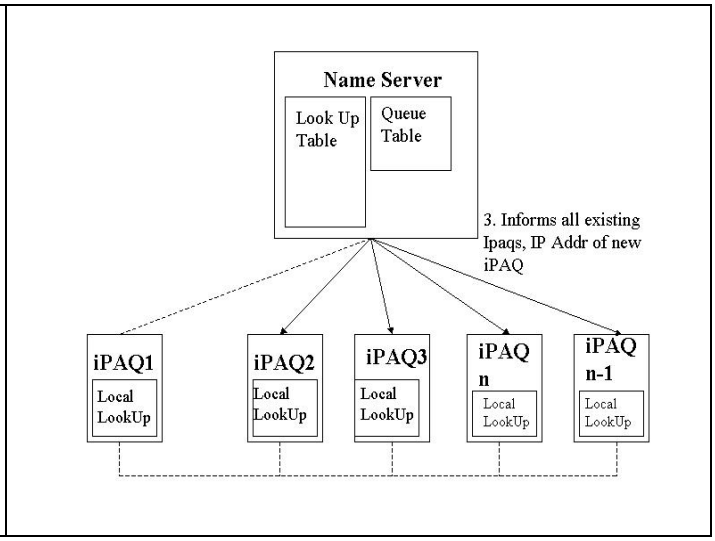

Figure 5: Updating all the other iPAQs local look up table

1. Referring to figure 4, we can see that whenever an iPAQ wants to connect to the group, it first sends the request to the server. It sends its ID number and its current IP address along with the request. The server first verifies if the iPAQ is allowed to join the group by matching the ID number sent by the iPAQ user to ID entries present in the server look up table. If the entry is not found then the server discards the request. If it is found then the server updates the corresponding currentIPaddr field by the IP address sent by the iPAQ with the request. This iPAQ now is in the "Active" state.
2. The server then sends the most recent view of the table to update the local copy of look up table maintained by the iPAQ. It sends all the ids of the other iPAQ in the group and their current IP address if they are active else a default of 000.000.000 indicating that it is "not active". Refer to figure 5. It also sends any messages that are stored in the Queue table for that iPAQ, thus making sure that it receives all missed messages. After sending all the missed messages it deletes it from the Queue table.
3. Whenever a new iPAQ joins the group, the server updates the respective corresponding currentIPaddr field in each iPAQ, which is connected to server at that time, with the IP address of the new iPAQ using multicast procedure.

- **Communication between iPAQs**

1. If one iPAQ, say iPAQ1 wants to communicate with another iPAQ, say iPAQ7 from the group, it can do so directly. Firstly iPAQ1 runs through its Local Look Up table and checks if iPAQ7 is in the active or not active stage.
2. If it is in the active stage, then it refers to iPAQ7's current IP address. Using direct communication message passing, it sends the message to iPAQ7. If for some reason iPAQ7 does not send an ACK back to iPAQ1, then it retries for n times. After n tries, iPAQ1 sends this message to the Name server.
3. The Name server stores this in the Queue table against the missed message for iPAQ7 and modifies the state of iPAQ7 to not active. It also informs all the other active iPAQs by multicasting the change.
4. If the state of iPAQ7 is not active, the sender iPAQ1 sends the message to the server, to be stored in the queue table. The server after storing the message sends an ACK to iPAQ7. This is an indirect form of communication.

- **Quit Procedure**

When an active iPAQ wants to quit the network, it sends the Name server a message indicating so. The Name server updates the state of that particular iPAQ to "not active". It sends a multicast to all the other

active iPAQs stating the quitting of this iPAQ, thus the Local Look Up are all updated. Finally the Name server sends an ACK to the quitting iPAQ.

- **Server Failure**

In case of failure of the Name Server, a secondary Back Up Server can be used, as seen in figure 3. This Back Up Server is the mirror image of the Name Server. As shown in figure 3, at a specific time interval, the Name Server sends a heartbeat to the back up server indicating that it is functioning properly. If the server fails to send a heart beat, the Back Up server takes over and acts like a Name Server. During this time if any iPAQ tries to connect to the Name Server, it will fail. After n retries it will connect to the Back Up server and join the group. As soon as the Name Server is up, it again sends a heartbeat to the Back Up Server and regains back the control.

- **iPAQ Failure**

For any reason if the iPAQ disconnects with the server without proper quitting routine, there might be discrepancy in the records of all the iPAQs. This can be over come by the message queue. If for any reason a message cannot reach the destined iPAQ after n retries, the sender iPAQ sends this message to the Name Server.

The Name Server stores this message in the message queue for that iPAQ and checks if the iPAQ is active or not. If it is active then it sends the message to the iPAQ. If not then it changes the state of this iPAQ in the Look Up table and notifies the other iPAQs. If yes then it sends the message to the iPAQ.

4.4 Analysis of the Proposed Architecture

We present an analytical evaluation of the Peer-to-Peer architecture with centralized control considering three factors: Number of Messages passed, Space (Storage) complexity and Response time.

4.4.1 Number of Messages Passed

Let us assume that there are 'n' iPAQs in the group, 1 Master Name Server, 1 Back-up Name server and 'k' Active iPAQs in the system. The number of messages passed is outlined below for different scenarios.

- **Start Up of an iPAQ**

In the *Normal case* when the Master Name server is working, the number of messages is given by: 1 message sent to the Name server + k messages to create the local Look Up table at the New iPAQ + m missed messages stored in Queue table for this New iPAQ + k IP address update messages to other Active iPAQs.

Total Number of Messages = (2k +m +1).

In case Message Bundling is supported, which means that multiple records can be bundled in a single message packet, the (k+m) individual messages to the New iPAQ reduces to $\lceil (k+m)/q \rceil$ where q is the number of records that can be bundled in one message. We can also use Multicasting to send the IP address message to other k iPAQs. This reduces the k IP update messages to a 1 Multicast Message.

The New Total number of messages = (1 + $\lceil (k+m)/q \rceil$ + 1^{Ψ}).

If the Master Name server fails, then the new iPAQ has to send an additional r_a retry messages to the failed Name server to ensure its failure and then send the start up message to the Backup Name server. The rest of the procedure is same as the normal case scenario.

Total Number of Messages = (2k + m +2 + r).

Total Number of Messages = (1 + r_a + 1 + $\lceil (k+m)/q \rceil$ + 1^{Ψ})

[With Message Bundling & Multicasting]

$^{\Psi}$ - Denotes a Multicast Message.

- **Communication between iPAQs**

When both the sender and receiving iPAQs are Active, there is 1 Communication Message sent to the receiver + 1 ACK message send back to the Source.

Total Number of Messages = 2.

Considering the case when the sender has a valid IP address of the receiver in its local look up table but the receiver suddenly goes out of range, then there is 1 communication message + 'r_b' retry messages to the receiver + 1 Queue Request Message to Name Server + (k-1) IP Update messages from Name server to other Active iPAQs to change status of out of range iPAQ to 'Not Active'

Total Number of Messages = ($1+ r_b + 1 + (k-1)$)

Total Number of Messages = ($1+ r_b + 1 + 1^{\Psi}$) [with Multicasting]

If the sender finds that the receiver is Not Active, then there is 1 Queue Request Message to Name Server + 1 ACK from Name Server + 1 Missed Message to the Receiver (this is accounted for in the Start up procedure).

Total Number of Messages = (1 + 1)

- **Quit Procedure of an iPAQ**

When an iPAQ quits, there is 1 Quit Message from iPAQ + 1 ACK Message from Name Server + (k-1) IP update messages to other Active iPAQs.

Total Number of Messages = (1 + 1 + (k-1))

Total Number of Messages = ($1 + 1 + 1^{\Psi}$) [with Multicasting]

From the above mathematical analysis we see that there is only a linear number of messages being passed in the system and is of the order O(k). Hence the architecture optimizes the number of messages passed to a considerable extent.

- **Adding Fault Tolerance to the System**

When an iPAQ suddenly goes out of range, the main problem arises when another iPAQ needs to communicate with it. The design of our framework supports this naturally by adding Retry messages and then Queuing messages with the Name server. There are no additional messages required.

For incorporating fault tolerance to the Name server, the Name server has to periodically send 'Heartbeat' message to the Backup Name server. Whenever the name server finds out that the state of an iPAQ changes, it has to propagate the change to the Back Up server.

Thus adding fault tolerance comes with an expense of additional message passing.

4.4.2 Space (Storage) Complexity

Our architecture uses 3 auxiliary data structures to achieve communication: Look Up table and Queue table on the Master Name server (also in the Backup Server) and a Local Look up table at each iPAQ.

- **Server Look up Table**

  There are n records one for each iPAQ. Let B the number of Bytes required to store each record.

  Total space required = n * B. [O(n)]

- **Server Queue Table**

  The Queue table maintains the two fields iPAQIDnumber and numOfMissedMgs for all the n iPAQs. Let B_1 be the number of bytes required to store these two fields. Let m be the number of records that have atleast 1 missed message. For a given record i with a missed message, let m_i be the number of missed messages. Let M be the number of bytes required to store a missed message.

  Total space required = $B_1 * n + \sum (M * m_i)$

i=1.. m

This is of order O(n +MK) where K is the total number of missed messages.

- **IPAQ Local Look Up Table**

  Total space is same as the Server Local Look up table. But this space is occupied in every iPAQ. Hence Total space for n iPAQs = n * B * n = n^2 * B.

4.4.3 Response time

Let U_1 be the Upper bound on the time required to send a message between an iPAQ and the Name Server, U_2 be the upper bound on the time to send a message between iPAQs and U_3 be the upper bound on the time to send a Multicast message by the Name Server to a group of iPAQs.

- **Start Up of an iPAQ**

  In case the Master Name Server works,

  Response time = $U_1 * \{1+ \lceil (k+m)/q \rceil \} + U_3$ (comes directly from Section 4.4.1)

  In case the Master Name Server has failed

  Response time = $U_1 * \{ 1 + r + 1 + \lceil (k+m)/q \rceil \} + U_3$

- **Communication between iPAQs**

  In case both the iPAQs are in the Active state, Response time = $U_2 * 2$

  In case the receiving iPAQ is out of range suddenly, then

  Response time = $U_2 * (1+r) + U_1 + U_3$

  In case the receiving iPAQ is Not Active, then Response time = $U_1 * 2$

- **Quit Procedure of an iPAQ**

  Response time = $U_1 * 2 + U_3$

  Note that all these response times are an Upper bound measure.

5 Features Supported by the Architecture

- **Collaboration**

The main objective of our architecture is to facilitate a group of people to collaborate in a wireless environment. Collaboration involves fixing up a common meeting time among two or more people in the group. The manner in which this is achieved is based on various voting schemes given below:

Equal voting: Vote of every member in the group has equal weight. In this case, there is a chance that a final decision cannot be reached at all, when there are equal number of 'Yes' and 'No' votes.

Priority voting: As the term indicates, a certain vote is given a priority over the others. When the members of the group receive a prioritized message, the meeting is scheduled.

Weighted voting: Here weights are assigned to votes from each sender. A decision is made giving more importance to messages with greatest weights.

- **Fault Tolerance**

A fault tolerant system is one, which is capable of handling system failures. In our design, there are two possible scenarios where the system might fail. One is the crashing of the Name Server and the other is iPAQs going offline abruptly without proper quitting. We have discussed this in section 4.3 under Server failure and iPAQ failure procedure.

- **Discovery**

Discovery means locating a resource. This is an important feature in a mobile environment where host gets abruptly disconnection from the network. In our framework since the location (IP address) of the iPAQ

constantly changes, we have designed Name Server. During the start up of the iPAQ, the Name Server aids in discovery of the location of every iPAQ in the group.

- **Reliability**

Reliability means that if one of the components in the system fail, then the system should not come to a halt. This is assured in our design by providing a Back Up Server in case of Name Server failure. Also the all missed messages are sent to iPAQ whenever it is active.

- **Security**

Security is a very important issue in a Mobile Environment. The system should be foolproof so that intruders cannot gain access to the contents of the Name Server. The System Admin provides each iPAQ user a unique user ID and password . The design may be extended to include additional encryption techniques to enforce added authentication and security to the system.

- **Scalability**

A system is said to be scalable if it can be extended to include any number of nodes and the performance of the system does not degrade. Our design is proposed for a small company, the design does not enforce any limitation on the number of iPAQs that can be involved, but there will be a bottleneck at the Name Server. As an extension to the architecture, we propose a hierarchical design. The design can consist of multiple local Name Servers each communicating with each other, to give an extensible view of the entire group and distributing the workload.

6 Conclusion

This paper discusses how a mobile working environment can be designed using the message passing paradigm. The detailed architecture for such a framework has been explained substantiating the key features required for group communication and collaboration using mobile devices. This kind of design supports the problem of dynamically changing IP addresses of the iPAQs. The procedures for operation of the framework in various scenarios have been presented and an analysis of the architecture has also been provided.

References

1. George Coulouris, Jean Dollimore, Tim Kindberg, “Distributed Systems Concepts and Design”, Addison Wesley Publishers Limited, Third Edition
2. David E Culler, Jaswinder Pal Singh with Anoop Gupta, “Parallel Computer Architecture, A Hardware Software Approach “
3. Silberschatz, Galvin “Operating Systems Concepts”, Fifth Edition
4. Tutorial on “Fundamentals of Interprocessor Communication”, http://www.cs.cf.ac.uk/Parallel/Year2/section3.html
5. Tutorial on Message Passing Overview, http://mrccs.man.ac.uk/hpctec/courses/IntroMP/IntroMP_2.html
6. Gregory R. Andrews , “Foundations of Multithreaded, Parallel, and Distributed Programming”, Addison-Wesley
7. Christopher Metz, IBM Corp., “Reliable Multicast: When Many Must Absolutely Positively Receive It”, IEEE Internet Computing July-August 1998
8. DNJ News : http://www.dnjonline.com/articles/mobility/iss25_mobility_tools.asp
9. “Windows CE: eMbedded Visual Tools 3.0 Provide a Flexible and Robust Development Environment” : http://msdn.microsoft.com/msdnmag/issues/01/01/CETools/CETools.asp
10. Nick Grattan , Marshall Brain “Windows CE 3.0: Application Programming” : http://vig.pearsoned.com/samplechapter/0130255920.pdf
11. Web Page on Windows CE Handhelds: http://www.openpocket.com/wince-pdas.htm
12. FOLDOC – Computing Dictionary web page: http://foldoc.doc.ic.ac.uk/foldoc/
13. What is – An IT specific Encyclopedia web page: http://whatis.techtarget.com/
14. Microsoft Web Page on ActiveSync 3.5: http://www.microsoft.com/mobile/pocketpc/downloads/activesync35.asp
15. Microsoft ActiveSync 3.5 Help Topic on “Introduction to ActiveSync”.