YCAB.NET CF:
COLLABORATION GROUPWARE FOR MOBILE DEVICES USING THE MICROSOFT.NET COMPACT FRAMEWORK

By

MIHIR P. PATEL

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

UNIVERSITY OF FLORIDA

2004

ACKNOWLEDGMENTS

I would like to sincerely thank Dr. Sumi Helal for his guidance and advice throughout the course of this project. I would also like to thank the other two members of my committee, Dr. Chris Jermaine and Dr. Baba Vemuri for their time and assistance.

I would also like to thank Harris Corporation for providing the nicely equipped Mobile Computing Lab. I would like to thank John Bowers, Denise Atteberry, Timothy Heffernan III and Kevin Austin for their help during my academic career.

Finally I would also like to thank my parents, sister and my fiancé for their love and support throughout my academic career.

TABLE OF CONTENTS

Page

LIST OF TABLES vi

LIST OF FIGURES vii

ABSTRACT ix

CHAPTER

1 INTRODUCTION 1

1.1 Thesis Objectives 3
1.2 Structure of the Thesis 3

2 OVERVIEW OF AD-HOC MOBILE COLLABORATION 5

2.1 Microsoft Net Meeting 3.0 5
2.2 YCab – JAVA 6
2.3 YCab.NET 7
2.4 Multihop Ad-hoc Instant Messaging 7

3 DESIGN AND ARCHITECTURE OF YCAB.NET CF 12

3.1 Connection Management Layer 13
3.2 Service Layer 14
3.2.1 Shared Whiteboard 16
3.2.2 Text chat service 19
3.2.3 Audio Chat Service 20
3.2.4 View peers 21
3.2.5 File Transfer Service 22
3.2.6 Shared Image Service 23
3.3 Client Management Layer 24
3.3.1 Client Manager 24
3.3.2 Client Frame 24
3.3.3 Message Router 25
3.3.3 Object Rebuilder 25
3.3.4 Packet Filter 26
3.4 Communication Management Layer 27

3.5 Data Transfer Layer 28
3.6 YCab.net CF Server 29
3.6.1 Central Storage of Messages 29
3.6.2 Main Server 30
3.6.3 Server Thread 31
3.6.4 Message Reader 31
3.6.5 Message Writer 31

4 DEVELOPMENT USING YCAB.NET CF 32

4.1 Design Specifications 33
4.2 Loading YCab.net CF 34
4.3 Creating a Custom Service 34
4.4 Setting the Namespace 35
4.5 Setting Inheritance 36
4.6 Implemented SharedColor Class 37
4.7 Handling User Input 37
4.8 Processing Incoming Messages 39
4.9 Compiling and Deploying 41
4.10 Output 42

5 CONCLUSION AND FUTURE WORK 43

5.1 YCab.net CF Server 44
5.2 Security 44
5.3 New Services 44

LIST OF REFERENCES 46

BIOGRAPHICAL SKETCH 48

LIST OF TABLES

Table page

2.1 Comparison of different collaborative software packages 9

2.2 Description of software components used in YCab.net CF 10

2.3 Description of hardware components used in YCab.net CF 11

LIST OF FIGURES

Figure page

2.1 Microsoft Net Meeting 3.0 5

2.2 YCab Java Sample Application 6

2.3 YCab.NET Application 7

2.4 Multihop Ad-hoc Instant Messenger 8

3.1 Layered architecture of YCab.net CF 13

3.2 Connection screen for sample application 14

3.3 Application without any services (left) and list of built-in services (right) 15

3.4 Shared Whiteboard service in YCab.net CF 16

3.5 Whiteboard as a 2 dimensional array (left) and derivation of x and y coordinates from the compressed form (right) 18

3.6 Text chat service in YCab.net CF 19

3.7 Voice chat service 20

3.8 File Transfer service 22

3.9 Shared Image service 23

3.10 Network mode selection during startup 28

4.1 Error detecting feature in Visual Studio.NET 2003 32

4.2 Adding a new class to the project 34

4.3 Design view of the SharedColor service 35

4.4 Setting proper namespaces 36

4.5 Setting proper inheritance 36

4.6 Constructor of SharedColor class....37
4.7 Adding an event handler....38
4.8 Implementation of the event handler for SharedColor service....39
4.9 Implementation of processMessage....40
4.10 Adding the new service to the Client Frame....41
4.11 SharedColor automatically added to the main menu....42
4.12 Initial screen of SharedColor (left), SharedColor screen after one event....42

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 Engineering

YCAB.NET CF: COLLABORATION GROUPWARE FOR MOBILE DEVICES USING THE MICROSOFT.NET COMPACT FRAMEWORK

By

Mihir P. Patel

August 2004

Chair: Abdelsalam (Sumi) Helal
Major Department: Computer and Information Science and Engineering

Today more and more people are using mobile devices such as handheld Personal Digital Assistants (PDAs), cellular phones, smart watches, etc. The phrase "mobile device" can be interpreted in many different ways. A laptop can be considered a mobile device, but a handheld PC or a cellular phone is a truly portable mobile device. These mobile devices are getting popular because they allow people to stay in touch with their friends, family, and business associates.

The new mobile devices support very specific collaborative applications, but there is a lack of an open standard or even a framework that would allow the development of collaborative applications for these small mobile devices. This thesis is focused on the development of one such framework called YCab.net CF – Wireless Collaboration using Microsoft .NET Compact Framework. YCab stands for Wireless Collaboration. The previous versions of YCab did not support development for mobile devices. This thesis proposes the newest version of the YCab project. This framework provides an API,

which can be used to develop applications for PDAs running Microsoft Pocket PC operating system and cellular phones running Microsoft SmartPhone operating system.

This thesis is not just about re-implementing the YCab architecture for small mobile devices, but about implementing a new customized version of YCab, which is optimized in terms of memory and processing power. Also the new YCab implementation needed to be compact enough to fit on the mobile devices, yet powerful and robust enough to support all the services that were included in the desktop versions of YCab , i.e., YCab Java and YCab.NET.

CHAPTER 1
INTRODUCTION

Mobile devices are growing in terms of popularity in today's market. The key feature that is pushing the sales in mobile devices is the wireless technology built-in the devices. With the wireless connectivity these mobile devices can be used for personal and business use to access the internet. The small mobile devices with wireless connectivity cab also be used to keep in touch with friends, family and business clients through collaborative applications.

Most of the software applications that are available for these devices are developed focusing on a particular need of the customer. For example, an application that allows a field engineer to take a picture of some construction site and send it to the main office is different then an application that allows a group of friends to keep in touch by sending text messages on their mobile devices. Thus all the collaborative applications that are available today are too specific, so there is need for a software package that allows a virtual space for collaboration on mobile devices. This virtual space can then be tailored to satisfy specific needs. Such a software package can not be an application, because an application can not be generic. Also an application can not be very flexible and extensible. This thesis discusses the design and architecture of a software package – YCab.net CF – that facilitates the creation of an ideal virtual space, which can be tailored to specific needs.

YCab.net CF is not an application, but is a basic framework that is implemented in form of an Application Programming Interface (API). YCab stands for Wireless

Collaboration. Lee [8] and Buszko [2] first developed such a framework – YCab Java – using Sun Microsystems's Java programming language. Applications developed using YCab Java can be used only on computers that had a Java Virtual Machine installed on them. Second framework was developed by Procopio [12] using the Microsoft .NET Framework. The framework which was called YCab.NET can be used to develop applications for any computer that runs any version of Microsoft Windows operating system. Both YCab Java and YCab.NET could be used to develop collaborative applications for desktop or laptops PCs, but there still was a need for a framework that would facilitate application development on handheld PCs and cellular phones. This thesis proposes the newest version of YCab project.

This thesis involved unique problems. With the devices getting smaller and smaller, so does their processing speed and internal memory. Also the programming environment gets limited. The complex programming structures and designs that are normally available on non-portable computers have to be built using only the simple structures available on these small mobile portable devices. For example the "Binary Serialization" (allows complex objects to be exchanged between different machines without any special effort) capability available on desktop PCs is not available on the handheld mobile devices, so special arrangements were made to exchange objects in YCab.net CF. Thus this thesis is not just about re-implementing the YCab architecture for small devices, but redesigning and adapting a new customized version of YCab which is optimized in terms of memory and processing power. Also the new YCab implementation needed to be compact enough to fit on the mobile devices, yet powerful and robust enough to support

all the services that were included in the desktop versions of YCab i.e., YCab Java and YCab.NET.

1.1 Thesis Objectives

The primary objective of this thesis is to design and implement a framework that will allow developers to create highly robust and optimized collaborative applications that can be used on PDAs running Microsoft Pocket PC operating system. Also develop built-in collaborative services such as shared whiteboard, text chat, shared files, shared image, voice chat and awareness. The project goals can be summarized by the following.

- Design and implement the YCab framework that could be supported on Pocket PCs
- Develop services that are available in the previous versions of YCab.
- Develop new services to enhance the user experience.
- Implement the framework using the features that are also supported on the cellular phones with SmartPhone operating system, so that the same framework can be used on cellular phones.
- Develop a sample application using YCab.net CF and document the API in a user-friendly format.

1.2 Structure of the Thesis

This thesis has a total of five chapters. Chapter 1 comprises of the introduction and the motivation behind developing YCab.net CF API. Also Chapter 1 presents the history of the YCab project and shows the roadmap that lead to the development of YCab.net CF.

Chapter 2 briefly describes some of the projects that focused on the issue of Ad-hoc mobile collaboration. Finally Chapter 2 describes the main technologies (hardware and software) used in YCab.net CF project.

Chapter 3 describes the design and architecture of the YCab.net CF framework. Further this chapter describes the implementation and function of each of the layers in the YCab.net CF framework.

Chapter 4 is targeted mainly towards the users of the YCab.net CF. This chapter step-by-step illustrates how a sample application can be built using the API.

Chapter 5 concludes this thesis and also mentions how YCab.net CF can be improved by implementing more user services. Also this chapter suggests some improvements in the current architecture.

CHAPTER 2
OVERVIEW OF AD-HOC MOBILE COLLABORATION

This section describes the projects that have focused on the issues of mobile collaboration.

1. Microsoft Net Meeting 3.0 at Microsoft Corporation.
2. YCab – Java at the University of Florida
3. YCab.NET – C# at the University of Florida
4. Multihop Ad Hoc Instant Messaging at Virginia Tech.

2.1 Microsoft Net Meeting 3.0

Net meeting is software developed by Microsoft Corporation. Net meeting is included in Windows Operating System 2000 and later by default. Net meeting is only supported by desktops running Windows operating system. Also it is based on a centralized architecture. The user has to first connect to a Microsoft net meeting server by using a registered user-id and password. Then the user can invite other people to join the collaboration by selecting their user-ids. Net meeting is not supported by any hand-held device (PDA, Smart phone, etc) other than a light weight laptop.

Figure 2.1 Microsoft Net Meeting 3.0

2.2 YCab – JAVA

YCab is a framework and API designed specifically for supporting rapid collaborative-application development. YCab API is written in Sun Microsystems's JAVA programming language. The application developed using YCab API allows mobile collaboration in Ad-Hoc environments. The application that uses YCab API runs using the Java Virtual Machine, so the application is only supported on the platforms that have a Java Virtual Machine. As almost no PDA or Smart phone as of today support the full version of the JAVA, such an application can not be used on any PDA or Smart phone. The YCab framework and API are thoroughly described in thesis by Lee [8] and Buszko [2], and is summarized in a journal paper by Lee, Buszko, and Helal [3]. The YCab project is therefore only briefly covered here. Below is a screen shot of a running application developed using YCab API.

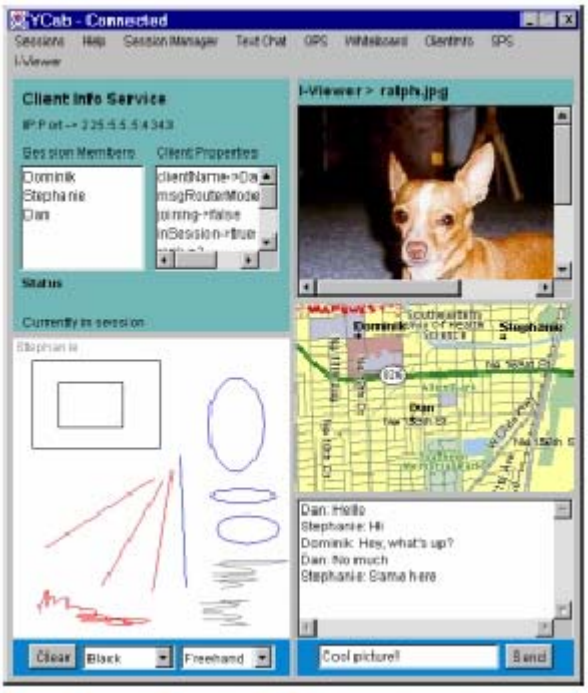


Figure 2.2 YCab Java Sample Application [2]

2.3 YCab.NET

The main goal of YCab.NET project was to port the original YCab from Java to Microsoft C# to utilize the managed and robust runtime and rapid application development environment of Microsoft .NET Framework. The long term goal of this project was to have YCab.NET run on PDAs running Windows CE, Pocket PC or SmartPhone operating systems. This project not only translated the code from Java to C# but also added numerous new features to the original YCab API. In YCab.NET the user interface was redesigned to give the application a commercial look. YCab.NET added streaming audio and improved support for file transfers. Over all the application developed with YCab.NET was very user-friendly.

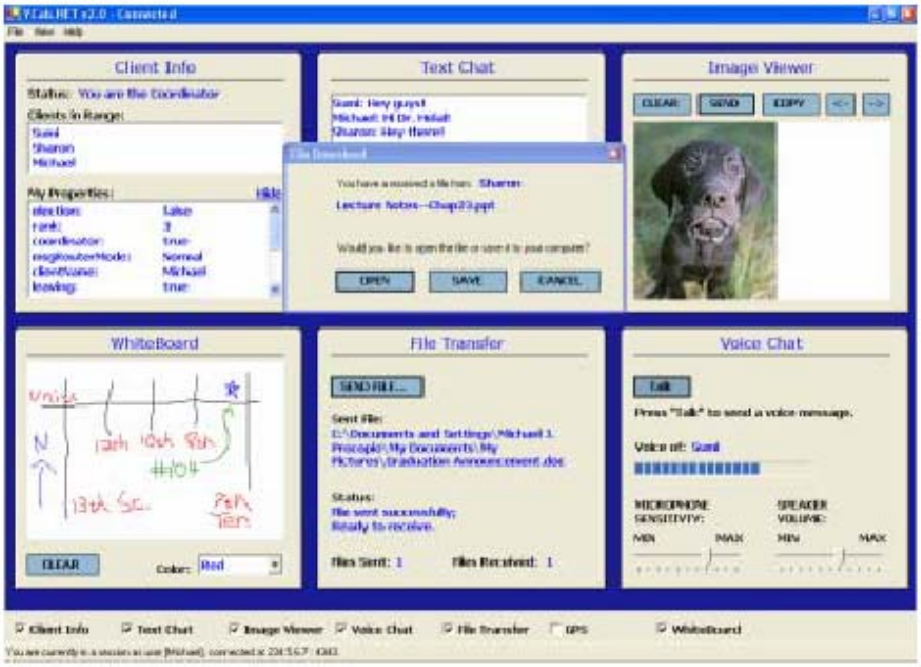


Figure 2.3 YCab.NET Application [12]

2.4 Multihop Ad-hoc Instant Messaging

This project was a part of the “Wireless Ad-Hoc messaging” project at Virginia Tech. This project unlike the ones mentioned above was supported by a PDA running

Microsoft Pocket PC operating system. The main goal of this project was to develop an application which allowed people with Pocket PCs to chat with each other in an ad-hoc environment. Whenever a user with a Pocket PC walks into the ad hoc environment, a "buddy" list similar to that in MSN Messenger's Instant Messaging pops up and the user can chat online with his/her buddies in the infrastructure-less ad hoc network. The project was implemented using Microsoft .Net Compact Framework® and C#® programming language, as well as Microsoft Smart Device Extension® for the .Net platform.

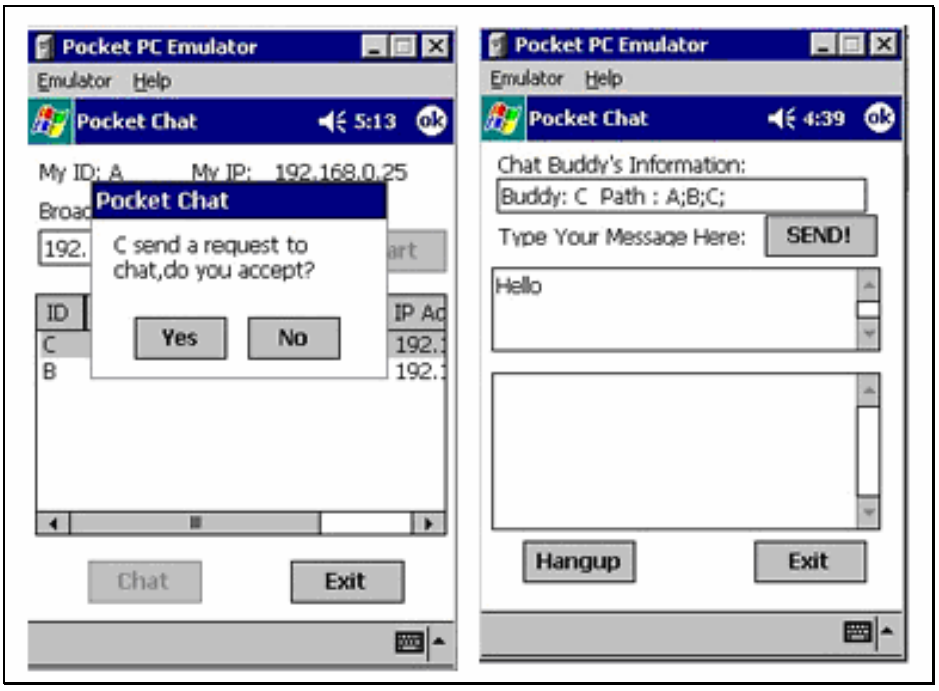


Figure 2.4 Multihop Ad-hoc Instant Messenger [4]

Table 2.1 Comparison of different collaborative software packages

	Microsoft Net Meeting	YCab Java	YCab.NET	Multihop Ad Hoc Instant Messaging
Supported Operating System	Microsoft Windows 95 or newer	Any	Microsoft Windows 95 or newer	Microsoft Pocket PC 2002 or newer
Hardware Requirements	Desktop PC or laptop PC	Desktop or laptop computer	Desktop PC or laptop PC	PDA with Pocket PC 2002 OS
Special Software Requirements	None	Sun Microsystems's Java Virtual Machine(JVM)	Microsoft .NET Framework	Microsoft .NET Compact Framework
Centralized architecture	YES	NO	NO	NO
Support for Ad Hoc networking environment	NO	YES	YES	YES
Text messaging	YES	YES	YES	YES
Multimedia collaboration	YES	YES	YES	NO
Extensible	NO	YES	YES	NO

Table 2.2 Description of software components used in YCab.net CF

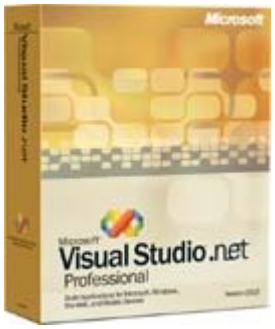
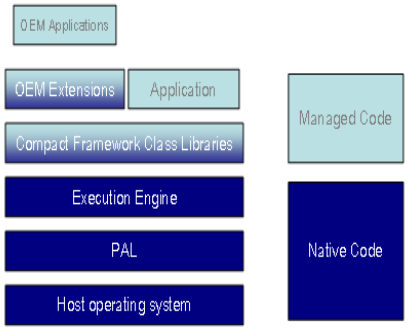
Component	Description
 **Visual Studio.net 2003**	Microsoft Visual Studio.net 2003 is a very rich Interactive Development Environment that allows the developers to edit, compile, debug and execute code all in one place. Also it has built-in support for Smart Device Extensions, which allows developers to develop applications for mobile devices such as PDAs and cellular phones.
 **.net Compact Framework**	The .NET Compact Framework consists of the base class libraries and has a few additional libraries that are specific to mobility and device development. The Framework runs on a high performance JIT Compiler. The Common Language Runtime is built from the ground up to be specific to the .NET Compact Framework so that it runs more efficiently on small devices.

Table 2.3 Description of hardware components used in YCab.net CF

Component	Description
 **Compaq Ipaq 3800**	Compaq Ipaq 3800 is a handheld PC running Microsoft Pocket PC 2002 operating system. It has built-in stereo speakers and a microphone. Its is powered by Intel 206 MHz StrongARM processor.
 **Dual expansion pack for Ipaq**	The dual-slot PC card expansion pack allows the user to use PC cards such as wireless network adapters, extra storage and battery packs. In this project the expansion pack was merely used for the wireless adapter.
 **Linksys Wireless Card ver 3.0**	The Linksys wireless network adapter ver 3.0 provides the capability of using the wireless network in infrastructure and ad-hoc mode. It also come with a software utility for Pocket PC OS.

CHAPTER 3
DESIGN AND ARCHITECTURE OF YCAB.NET CF

YCab.NET CF is a framework and API that allows software developers to create collaborative applications that harnesses the power of Microsoft .NET Compact Framework (.NET CF). These applications can be used on any device that supports .NET CF. This chapter will discuss the implementation of the framework and the sample application.

An application developed using YCab.NET CF has different services. A "Service" in YCab.NET CF is a custom built application that is a part of the main application. All services together make the application a true collaborative space. From a .NET programmer's perspective a service in YCab.NET CF is a subclass of System.Windows.Forms.Form. The class "Service" is implemented in the API. The service class contains all the common GUI components that all the services need like the main menu that contains names of all services and the "exit" option to close the application. This base class also contains the button that will display the contents of the help file associated with the service. The Service class provides a high-level abstraction to the application developer by hiding all the low level implementation details and the calls to the lower levels in the API. The API can be divided into a layered structure as shown in figure 3.1. The layers that are enclosed by the dotted line form the core of the architecture. The service layer that lies outside of the dotted line is the application layer. The service layer is formed of customizable components called services. There are built-in services in this layer to which new custom services can also be added.

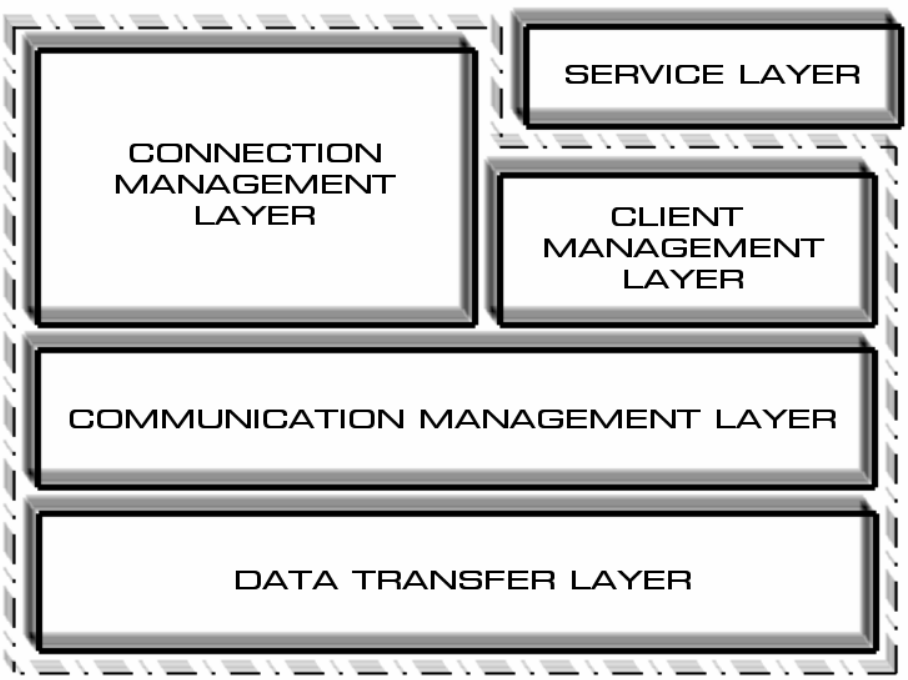


Figure 3.1 Layered architecture of YCab.net CF

3.1 Connection Management Layer

This is the very first layer in the YCab.net CF architecture. This layer has components that deal only with getting the user to connect into the virtual collaboration space. The main class that handles most of the work related to the connection is the Connection Manager. When the sample application provided with YCab.net CF API starts, the screen shown in Figure 3.2 is displayed. Once the user clicks on the “Connect” button, all the information the user fills out in the form is sent to the connection manager. The connection manager at this point starts a timer that times out after certain seconds. Then the connection manager sends this information to all the collaborators currently present in the collaboration.

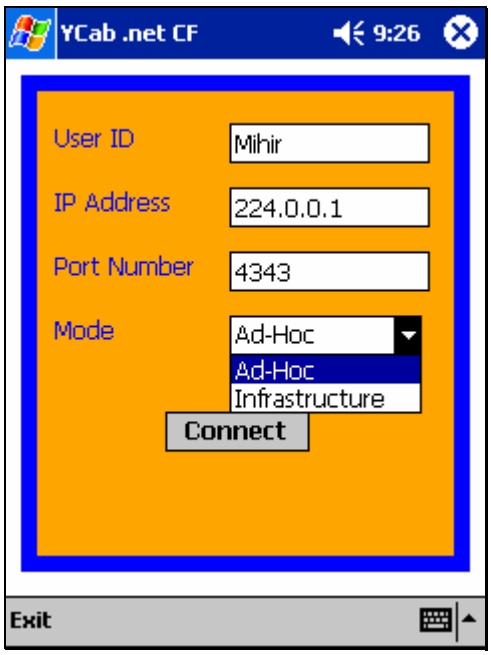


Figure 3.2 Connection screen for sample application

If any of the collaborators is using the same userid, then that particular collaborator will send a message to the new user who is trying to get connected. When the new user receives such a message, the connection manager will display the above shown screen again with the message "Userid in use, please select another userid". If the timer that the connection manager started times out and the new user does not receive any message, then the control of the application is forwarded to the lower layer in the API i.e., the service layer. Once successfully connected, all the available services are displayed in the main menu of the application. The user can chose any service and start collaborating with other users.

3.2 Service Layer

This layer contains all the services. There are two major types of services. The first one is the system level services. The system level services are the ones that are absolutely needed by the application to run effectively. An example of such a service is the

connection service that allows the user to connect to his/her peers. Another example is the awareness service; this service allows the user to view the list of peers involved in the collaboration. If this service stops working then, there is no way for the user to know if his/her peer is not replying or he/she is not in the wireless network range(i.e., has lost connection). The other major type of service is the user level service. These are the services that the user interacts with the most. Examples of this type of services are the text chat service, shared image service, file transfer service, etc. These services improve the value of the application by providing different ways to collaborate with peers. All the services inherit from the base class "Service". This class abstracts the implementation details that involve communicating with the lower level classes in the API such as the Client Manager, Message Router, Packet Filter, etc. The following figure shows the application that is started without initializing any of the services.

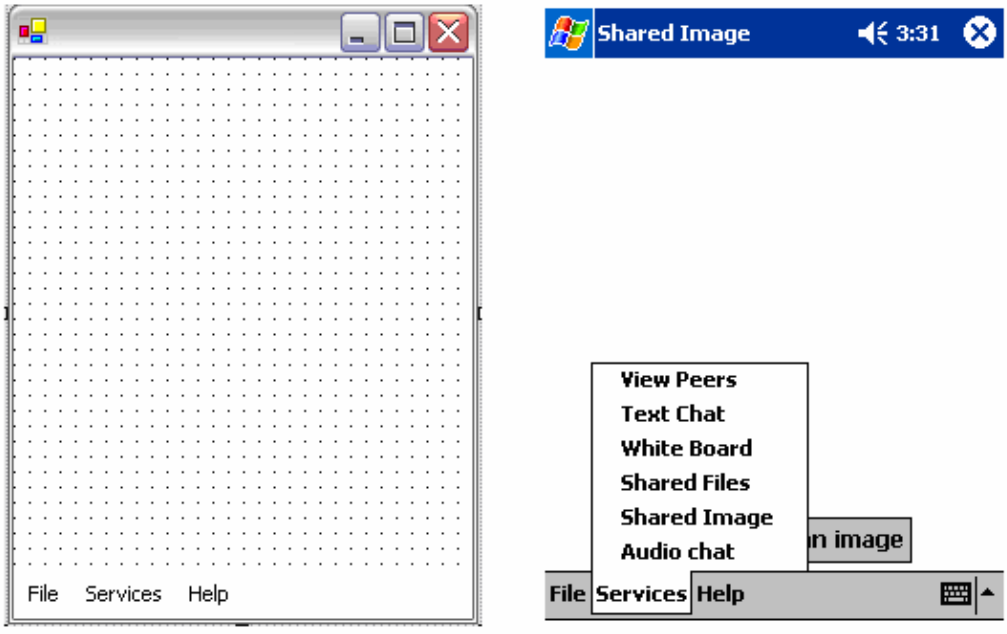


Figure 3.3 Application without any services (left) and list of built-in services (right)

There are a number of built-in services in the YCAB.NET CF API which are described in the following sections. The service layer is formed of the following services.

3.2.1 Shared Whiteboard

Let us say we have Bob, John and Jane in a collaboration space created by an application developed from YCab.NET CF and this application has the shared whiteboard service. This service provides a virtual whiteboard, where each of the collaborators Bob, John and Jane are allowed to draw, erase and clear the whiteboard. Let us say Bob chooses the color of the pen from the drop down menu and starts drawing on the whiteboard. The moment he pushes down his stylus on the Pocket PC screen, the service starts capturing the movement of the stylus in form of x and y coordinates. Once Bob is finished drawing and lifts the stylus off the screen, a message containing the captured x and y coordinates and the color of the pen Bob used is sent to John and Jane. Once the message is received on John and Jane's side, the service will use the x and y coordinates and the color of the

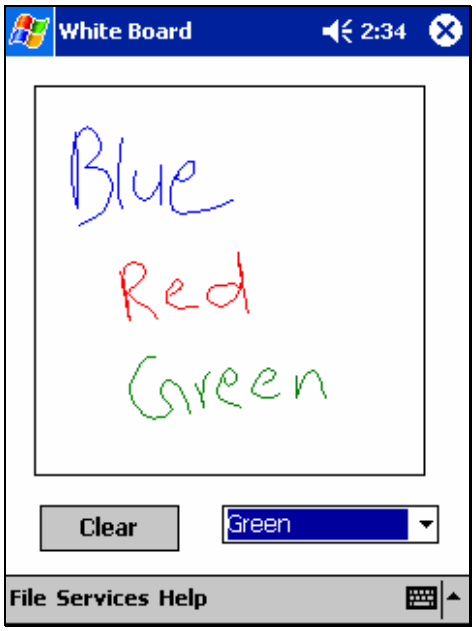


Figure 3.4 Shared Whiteboard service in YCab.net CF

pen specified in the message to recreate the drawing that Bob drew on his Pocket PC. Now let us say Jane wants to add something to Bob's drawing, she can pick a different colored pen and draw on her Pocket PC. Once again the service will send the x and y coordinates that forms Jane's drawing and the color of her pen and sends them to Bob and John. Now let us say John wants to clear the whiteboard and start over, he can use the button labeled "Clear" and once again a message will be sent. But this time a special message will be sent. Once the message is received on Jane and Bob's Pocket PC, the service will identify the special message and clear the whiteboard. Thus this service provides a truly shared whiteboard, where the collaborators can express their ideas by drawing on the whiteboard just as in real meetings.

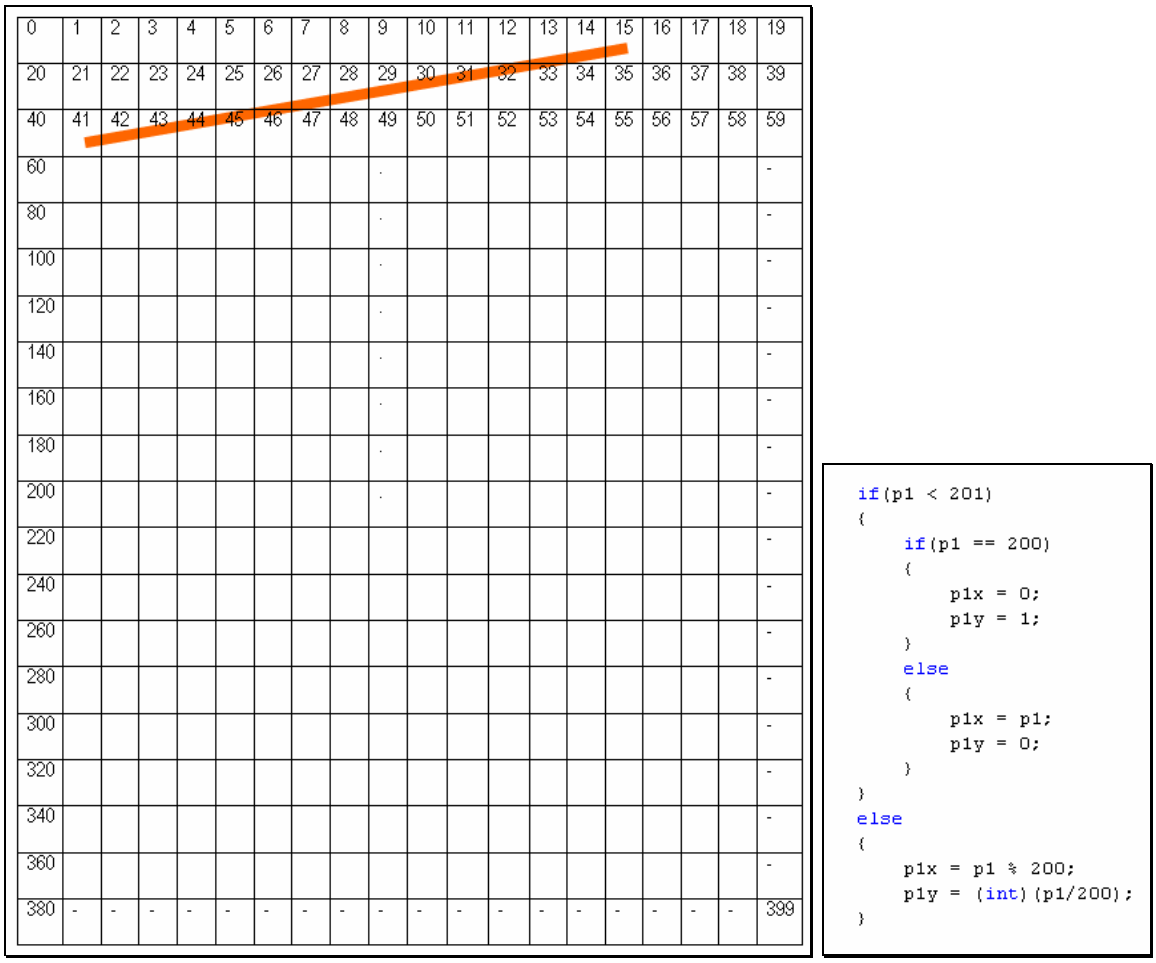
To have a real shared whiteboard, instead of sending the state of the whiteboard after every change to all the peers, this service only sends the change in the state of the whiteboard. To further optimize the size of the message sent after every change the following steps were taken. The x and y coordinates of the position of the stylus are of type "int" in C#, which takes 4 bytes each. So to send one line which is formed of two points, we need to send 4 integers which would take 16 bytes total. The drawing area on the screen is limited to 200 X 200 pixels, where pixel 0 is in the top left of the drawing area. Now the drawing area can be mapped into a two dimensional array with x and y coordinate as the indices in the array. Further a two dimensional array can be represented as a one dimensional array. The following example illustrates the optimization of the message sent by the "Shared Whiteboard" service. Let us say that the table shown in the next figure represents the whiteboard. Now each cell in the table is a pixel in the

whiteboard, so the table represents a whiteboard of size 20 X 20 pixels. The orange line starts from the point (1,2) and ends at (15,0)

a(1,2) = 2 * 20 + 1 = 41

b(15,0) = 0* 20 + 15 = 15

Thus, the orange line can be represented with two integers 41 and 15 instead of two points with two integers for each point. On the other side the two points are derived using the following algorithm where p1 represents the point that is being derived.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
60									.										-
80									.										-
100									.										-
120									.										-
140									.										-
160									.										-
180									.										-
200									.										-
220																			-
240																			-
260																			-
280																			-
300																			-
320																			-
340																			-
360																			-
380	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	399

```
if(p1 < 201)
{
    if(p1 == 200)
    {
        p1x = 0;
        p1y = 1;
    }
    else
    {
        p1x = p1;
        p1y = 0;
    }
}
else
{
    p1x = p1 % 200;
    p1y = (int)(p1/200);
}
```

Figure 3.5 Whiteboard as a 2 dimensional array (left) and derivation of x and y coordinates from the compressed form (right)

3.2.2 Text chat service

Text chat service allows the collaborators to send text messages to each other. First the user enters the message in a text box using any of the text input mechanisms available on the mobile device, and then clicks on the send button. Once the user clicks the send button, an array of bytes is instantiated with the American Standard Code for Information Interchange (ASCII) encoded text message as its content. The message array along with the user's userid is then forwarded to the user manager and then to the lower levels in the API and finally into the wireless network. Upon receipt of such a message from the network, the text-chat service will first decode the byte array called message using the ASCII standards and display it in the text-box labeled "History" in figure 3.6 along with the sender's userid. This service could be easily extended to include private message. For example, if there are five collaborators and user1 wants to send a private message to user2, then user1 can write the message that starts with special character sequence

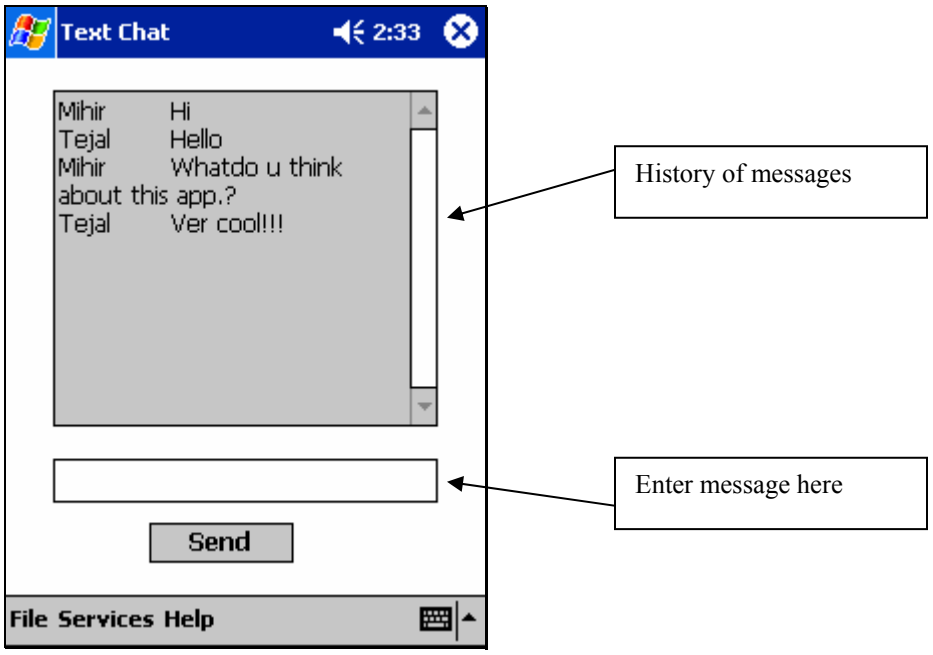


Figure 3.6 Text chat service in YCab.net CF

followed by the receiver's userid/s. Upon receipt of such a message if the receiver of the message is not same as the user's name, then the message is not displayed in the "History of messages" (see figure 3.6) and thus the message stays confidential.

3.2.3 Audio Chat Service

This service allows the collaborators to send and receive voice messages to/from each other. When the user clicks on the button labeled "Talk," voice recording starts. At this time, the user can start talking. Also right after the talk button is clicked, a floating GUI component is displayed, which has a button labeled "OK." Once the user is done talking, he/she can click on the "OK" button on the floating GUI component. Once the "OK" button is clicked the floating GUI component is hidden. All the data corresponding to the voice message are then used to generate a file with extension "wav." WAVE file format is very simple. WAVE files contain the raw PCM - Pulse Code Modulated - audio data. This file format does not use any compression. Once the wave file is created, it is sent to the lower layers of the API for transmission. Once the file is

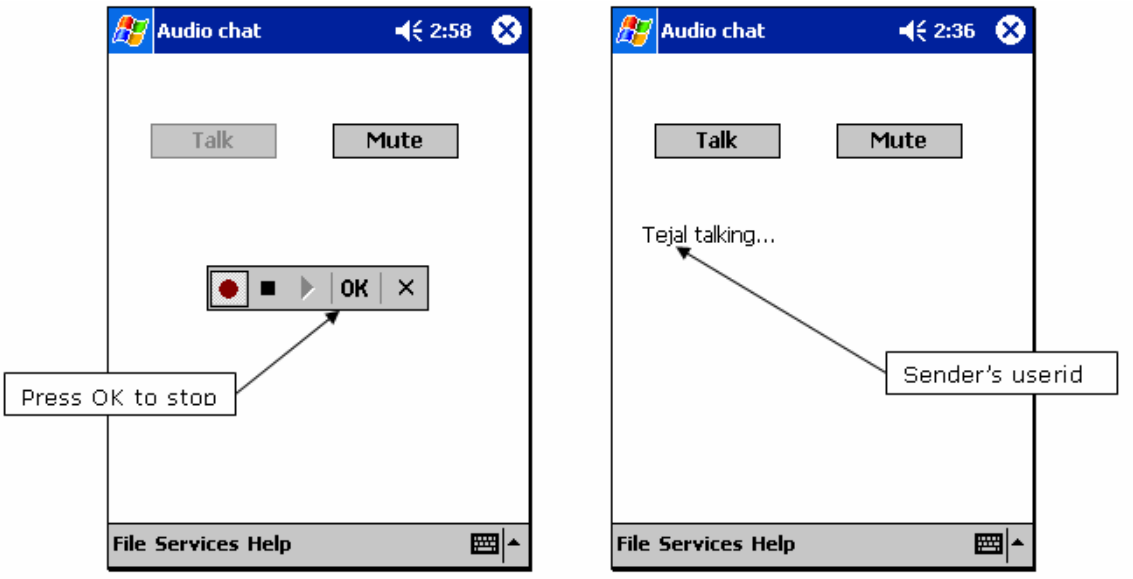


Figure 3.7 Voice chat service

received on the other user's application, the sound file is played. A text message is also displayed with the sender's userid, so that everyone can see who sent the voice message. Whenever the recording or playback of a voice message is completed the files containing the message is always deleted to preserve memory on the user's machine.

3.2.4 View peers

This service allows every collaborator to view the userid of every other collaborator. Also this service can be easily extended to display other information for each user i.e., first name, last name, a head-shot, etc. This service at all times maintains a hash-table data structure. Each entry in this hash-table is a pair of userid and a timestamp. The timestamp is just an integer which is initialized to zero in the beginning. This service periodically sends out a special message called a ping-message. This message contains all the information about the user i.e., userid, first name, last name, etc. Once this information is received on the other users' mobile device, this service checks if sender's information is already present in the hash-table. If the info is not present, then it is added to the hash-table along with the sender's userid with the current timestamp. If the information is already present in the hash-table, then the timestamp associated with the sender's userid is updated. Every time a message is received, the timestamp is increased by 1. After a certain timeout period, the hash-table is cleaned up by removing userids that have a very old timestamp. A "Very old timestamp" is defined by an integer constant which is set to 50. Thus after receiving 50 messages, the hash-table is cleaned up to reflect the absence of collaborators who got disconnected due to network unavailability or left the collaboration space voluntarily.

3.2.5 File Transfer Service

The File-Transfer service allows the collaborators to share files. This service provides each user the capability to send out a file to other peers, and also to receive files from them. To send a file, the user should click on the button labeled “Click to send a file”, which opens up the user’s “My Documents” folder. Now the user can select the file that he/she wants to send by just clicking on the file name. Once the user selects the file, the file is read into a byte array and sent down to the lower layers of the YCAB.NET CF API for transmission. Once the file is received on all the other user’s mobile device, it is added to the list of received file as shown in figure 3.8. This service can be extended by allowing the user to view the file from the received files by clicking on it. Also one of the limitations of this service is that the files that are to be sent out should be under the “My Documents” folder. This limitation is inherited from the underlying .NET Compact Framework.

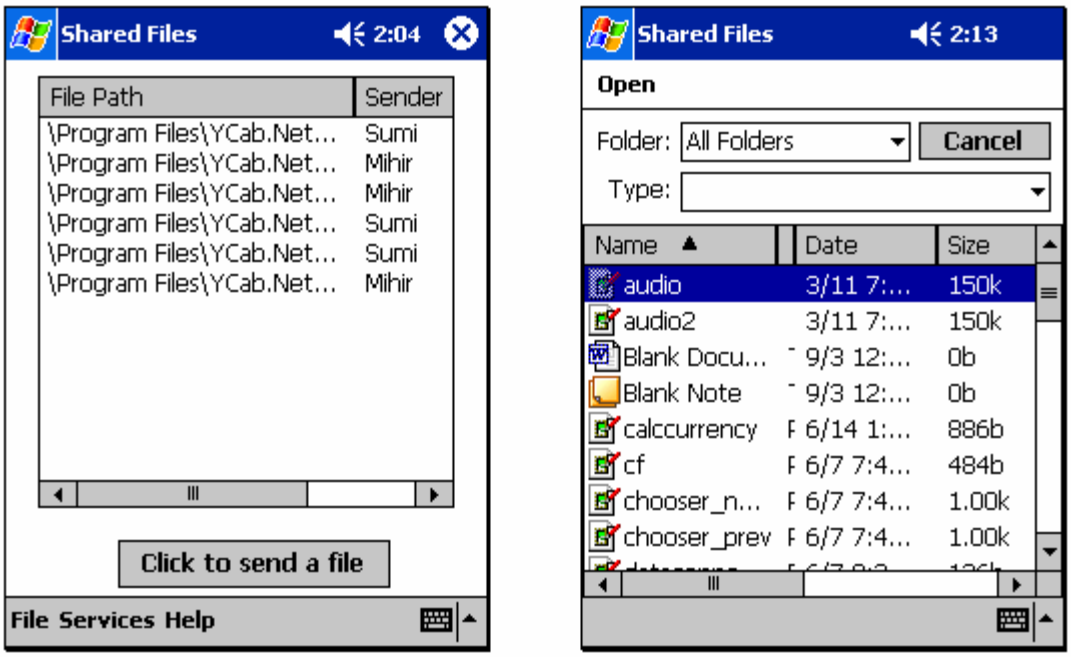


Figure 3.8 File Transfer service

3.2.6 Shared Image Service

The Shared-Image service allows the collaborators to share an image. Any of the collaborators can post an image on this virtually shared space. At any given time, all the collaborators will see the exact same image. To post an image on this virtual “wall”, the user will first click on the button labeled “Click to send an image”. Once the user clicks on the button, the user will be asked to choose an image from the list of the images stored under the “My Documents” folder on the user’s mobile device. The user can select the image by clicking on it. As soon as the user selects the image, the image file will be read into a byte array and sent down to the lower layers of the API for transmission. Once the file is received on the other user’s mobile device, the image will be displayed on the screen. If an invalid image file or corrupted file is received, then an error message will be displayed letting the user know the name of the sender who sent the invalid image file.

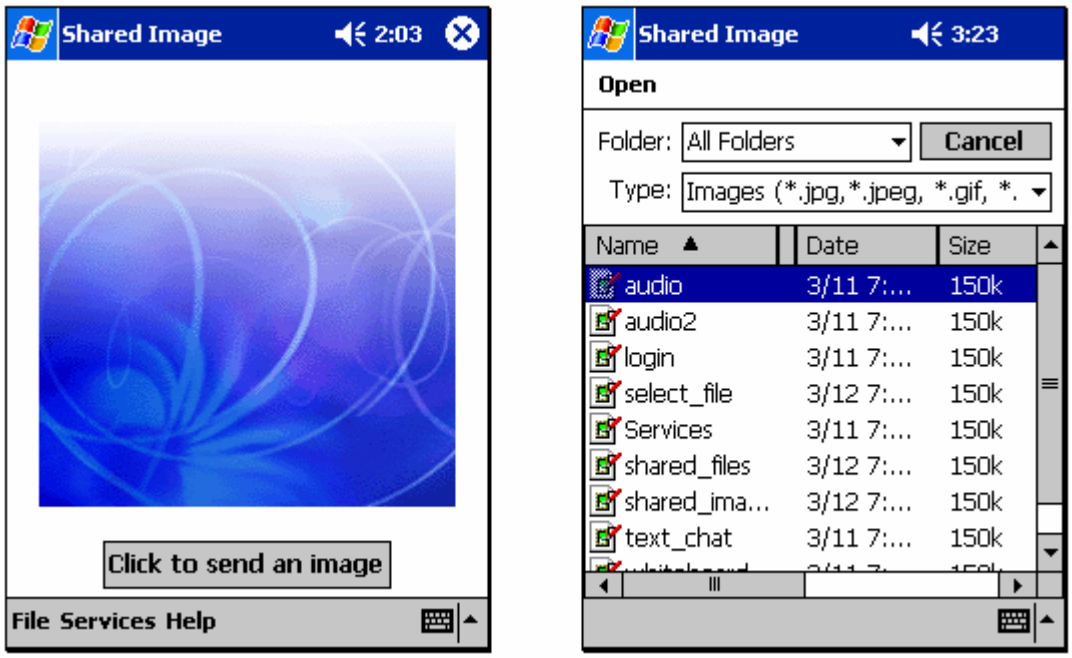


Figure 3.9 Shared Image service

This service is very close to the File-Transfer service in terms of implementation. Just like the File-Transfer service, this service also has a limitation that the image files to be sent out should reside under “My Documents” folder on the sender’s mobile device.

3.3 Client Management Layer

As the name suggests, this layer is responsible for managing the user/client’s state in collaboration. This layer directly interacts with the Service layer. This layer is made up of the following classes.

3.3.1 Client Manager

This class is responsible for smooth operation of all the services. The user manager provides helpful information to all the services such as the name of the user and list of names of all the peers that the user is in collaboration with. Also the client manager is directly in contact with every service through the base “Service” class. The client manager forwards the message from all the services to the Packet Filter object. Once the Packet Filter object splits the message into smaller messages if necessary, the client manager forwards the messages one level down to the Communication Manager. The client manager is also responsible for shutting down all the services and the Communication Manager when the application exits. In short the client manager manages all the utility classes in the API i.e., the Packet filter, Message Router, all the services, the Communication Manager and the client identity.

3.3.2 Client Frame

This class acts as a container for all the services. All the services – system level and user level – are contained in the Client Frame. This class keeps a record of all the registered services in the application. The client frame provides the interface to the application developer to add custom services. The client frame can also be used to

initialize the system-level services that are absolutely required for the application to run ex. Ping service, identity protection service, etc. Whenever a new service is added to the client frame, the client frame makes sure that the new service is added to the main menu of the application. When the user clicks on the main menu of the application and selects a service, it's the client frame that actually displays the appropriate service on the screen. Thus this class is also responsible for allowing the user to switch between different services through the main menu of the application.

3.3.3 Message Router

The instance of this class runs as a separate thread from the main application. Message Router is managed by the Client Manager. The Client Manager enables routing at the initialization of the application and disables it when the application terminates. The Message Router is responsible for routing the messages that it gets from the lower layers in the API to their destination services, which are one layer up. For every incoming message the Message Router first checks the header of the message to figure out which service is the recipient of the message. The Message Router is also responsible for forwarding packets to the packet filter if the message is marked as a partial message.

3.3.3 Object Rebuilder

The Object Rebuilder runs as a separate thread from the main application. It is started by the Packet Filter whenever the Packet Filter receives all the parts of a partial message. The Object Rebuilder uses the partial messages and the sequence number associated with each message to construct the original message. Once the message is constructed, it forwards the message to the message router. For every message that is broken into partial messages, there is one Object Rebuilder. The Object Rebuilder thread dies as soon as it finishes the construction of the original message.

3.3.4 Packet Filter

The Packet Filter is managed by the Client Manager. The Packet Filter is initialized when the application starts up. The main job of the Packet Filter is to make sure that the messages that are sent by the service layer are not bigger than the maximum size allowed by the communication manager who is one layer below. Whenever the service layer sends a message to the client manager to transmit out into the network, the client manager first sends the message to the packet filter. The packet filter checks the size of the message to make sure that it is not bigger than the maximum allowed size. If the packet is larger than the allowed size, then the packet filter constructs partial messages, where the payload of each message is a part of the original message. All the partial messages are marked with a random sequence number which is common to all the partial messages and a unique multipacket number which will later be used to put the packets in the right order to reconstruct the original message. Also all the partial messages are marked as "Partial Messages" so that they are not forwarded to the service layer directly. Once the original message is fragmented into smaller messages, the packet filter forwards this array of new smaller messages to the client manager which is then forwarded to the lower layer for transmission.

Whenever the message router receives a message/packet that is marked as a partial message, instead of forwarding the message to the service layer, it will forward the message to the packet filter. The packet filter at all times maintains a hashtable of partial messages. Each entry in the hashtable is formed of a pair of sequence number and an array of messages. Whenever the message router forwards a packet to the packet filter, the packet filter first gets the sequence number of the packet. Then it will check the hashtable to see if it already has packets with the same sequence number in the hashtable.

If it finds the sequence number in the hashtable, then it will append the packet in the array associated with that particular sequence number. If that particular sequence number is not present in the hashtable, then it will add a new entry for that sequence number. The array associated with that sequence number will be empty. Now every time the packet filter adds a partial message to any of the arrays in the hashtable, it will check if the array is full. If the array if full, then it initializes a new Object Rebuilder and assigns it the full array. Also at this point it will delete the entry associated with that array from the hashtable.

3.4 Communication Management Layer

This layer of the API abstracts the details of communication with other collaborators in the wireless network. The application made using the YCAB.NET CF API, can be used in an ad-hoc mode where there is no physical infrastructure available to support connection between peers. Also the same application can be used in an infrastructure mode where there is a structural wireless network established by using wireless access points and/or wireless routers. This layer provides a unique interface to the layers above, so that the services layer or the client management layer does not have to know which medium (ad-hoc or infrastructure) is used for communication. The only class that resides on this layer and provides this high level of abstraction to the layers above is the “Communications Manager” class. This layer directly interacts with the lowest layer in the API namely the “Data Transfer Layer” and the layer above the current layer namely the “Client Management Layer”. When the application is initialized, the user is asked to chose the medium of communication as seen in the figure below.

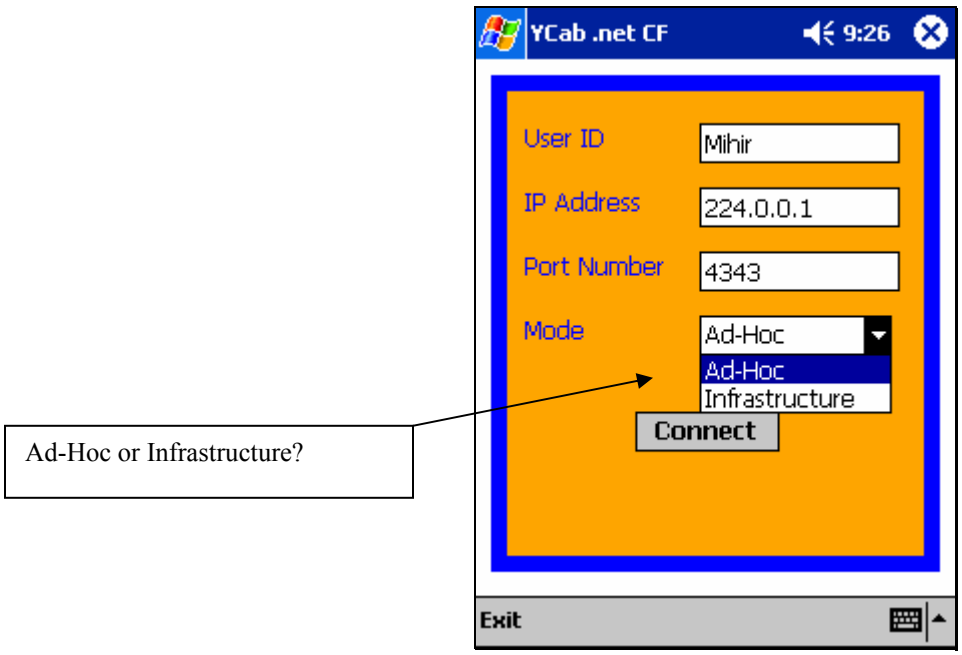


Figure 3.10 Network mode selection during startup

If the user selects the Ad-Hoc mode, then the communication is done using the User Datagram Protocol – UDP. If the user selects the infrastructure mode then the communication is done using the Transmission Control Protocol - TCP. Either way all the classes above this layer never have to know as to what mode is used to exchange messages. The communications manager manages the two main components that reside in the "Data Transfer Layer". These components are described in the following section.

3.5 Data Transfer Layer

This layer consist of two main components namely the "Receiver" and the "Sender". Both of these components are initialized by the communications manager. Depending on the communication mode the user chooses during the initialization of the application, the communications manager will initialize the receiver and the sender. If the mode selected is ad-hoc, then the receiver and sender will be initialized in such a manner that they will listen and send UDP messages to/from the socket. If the selected mode is

infrastructure, then the communications manager will initialize the receiver and the sender to listen and send TCP messages to/from the socket.

When the selected mode of communication is infrastructure, then there is a central server to which all the collaborators have to connect in order to communicate with each other. There is a server included in the YCAB.NET CF API. The server is implemented in Java so it can be run from any operating system i.e., Solaris, Linux, Windows, etc. The following section will give brief description of the server.

3.6 YCab.net CF Server

The server plays an important role in providing the communication level abstraction to all the layers in the API. All the components of different layers of the API described above function in the exact same manner when the server is used for communication and even when communication is done directly in an ad-hoc network environment. The server acts as message router. The server can be initialized by passing the port number to be used as an argument. The only requirement for the server to run properly is a valid public IP address, one open port and the Sun Microsystems's Java Virtual Machine. To initialize the server, the following command should be used at the command prompt.

prompt %> java Server x

where x is a valid open port number.

The server has following main components.

3.6.1 Central Storage of Messages

There is a class called "Messages" in the YCab.net CF server. This class acts as a central storage. All the incoming messages are stored in a queue. The first message to arrive is the first message to depart from the queue. This allows the proper ordering of

messages. Every time a client sends a message to the server, the server forwards that message to this central storage. The message is put at the end of the queue. The central storage also keeps track of currently active number of clients. This storage directly works with two other components of the server i.e., Message Reader and Message Writer. This storage facility has an algorithm to make sure that duplicate messages are not sent to the same client. When a message writer requests a new message, the central storage will increment a counter. When that counter becomes equal to the number of clients, that message is permanently deleted from the queue and the temporary counter is again set to zero. Whenever a new client connects with the main server, the number of clients is incremented by one. Also whenever a client disconnects from the server, the message reader or the message writer will notify the central storage, so the number of clients will be decremented by one.

3.6.2 Main Server

The main server listens for connections on the port number specified as the command line argument. But before it starts listening, it creates an instance of "Messages". This object acts as the central storage of messages. Now whenever a client tries to connect to the server, the server creates a new thread to handle the client, so for each client, there is a separate server thread. Once the server creates a new thread, that thread is assigned the current client. This new server thread is also given a reference to the central storage of messages. The main server again goes back to listening. The call to the listen method in the main server is blocking, so the server is not using any resources while it is waiting for connections.

3.6.3 Server Thread

Whenever a new client tries to connect to the YCab.net CF server, the main server creates an instance of the “Server Thread”. Also the main server gives the server thread a client to handle and the reference to the central storage of messages. Now the server thread gets the input stream and output stream of the client socket that is used for communication between the main server and the client. The server thread instantiates two new objects i.e., Message Reader and Message Writer and passes them the input stream and output stream respectively. The Message Reader and Message Writer both run as separate threads. The server threads’ job is done at this point.

3.6.4 Message Reader

The message reader is responsible for getting all the incoming messages from the client. Whenever the message reader gets a message from the client, it forwards the message to the central storage. Once the message is forwarded, the message reader goes back to reading.

3.6.5 Message Writer

The message writer is responsible for sending all the messages stored in the central storage to its client. The message writer sleeps until there is a new message to be sent. When there is new message available in the central storage, it will forward that message to its client and go back to sleep again.

CHAPTER 4
DEVELOPMENT USING YCAB.NET CF

The YCab.net CF API was developed using C# (pronounced C Sharp) programming language. C# can be written in any text editor. C# does not require any special Interactive Development Environment (IDE), however the entire YCab.net CF API was developed using the Microsoft Visual Studio. NET 2003 (VS) IDE because VS provides a very proactive development environment that supports editing, compiling and debugging programs written in over 17 different languages. Note here that the development environment is VS 2003 and not VS 2002. This is because only VS 2003 and newer versions of VS will support development for mobile devices. Some of the main features of VS 2003 are as follows.

- Visual form designer – VS allows developers to design Windows Forms by just dragging and dropping controls such as buttons, text-boxes, labels etc.
- VS checks for syntactic errors while the developer is coding. In figure 4.1, there is an extra closing parenthesis at the end. VS will immediately detect it and notify the developer by showing the red line where there is an error and also a hint as to what could be the reason for the error.

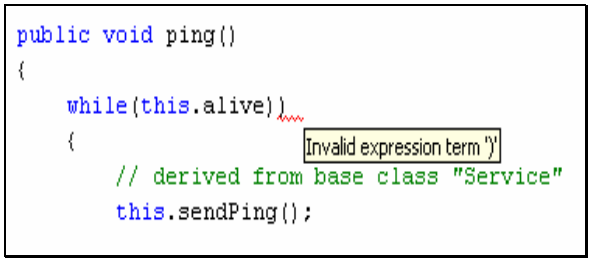


Figure 4.1 Error detecting feature in Visual Studio.NET 2003

- VS has a very robust built-in debugger, which lets you insert breakpoints, lets you execute the code line by line and also lets you set a “watch” on variables, that notifies the developer when the value of a certain variable changes. There are a lot

of advanced debugging options available in VS, but it is beyond the scope of this thesis to describe them.

- VS has built in support for Smart Device Extensions, which is a Microsoft product that allows developer of C# and Visual Basic.NET to develop application for mobile devices.

For developing applications using the YCab.net CF API, the developer is not required to use VS, but it is highly recommended that he/she does because of the ease in development process. VS facilitates easy and rapid application development.

Sample Application

This section provides a step by step instruction on how to create a simple application using the YCab.net CF API. The API provides a very intuitive interface for the developers. The application will connect any number of clients using the TCP/IP or UDP protocols, but the developer is not required to have any networking knowledge. The developer is only required to be familiar with C# programming language and some what familiar with Visual Studio .NET 2003. Also the developer is required to know the interface of the YCab.net CF API which is well documented in the API documentation provided with the YCab.net CF API.

4.1 Design Specifications

This sample application will have a screen with a colored background, and a button labeled “Next”. Also there is a sequence of colors defined in the application. When the user will click on the “Next” button, the background color of the screen will change to the next color in the predefined sequence. The next button can be clicked however many times and by any of the collaborators. At any given time all the collaborators will see the exact same color on the screens of their mobile devices.

4.2 Loading YCab.net CF

Start up Visual Studio .NET 2003. From the main menu of VS, select the option to open an existing project. Browse to the directory where YCab.net CF is installed and select the main project file. If there is no error displayed then, YCab.net CF has successfully loaded at this point.

4.3 Creating a Custom Service

Now from the project menu, select "Add Class", then name the class SharedColor as shown in the following figure.

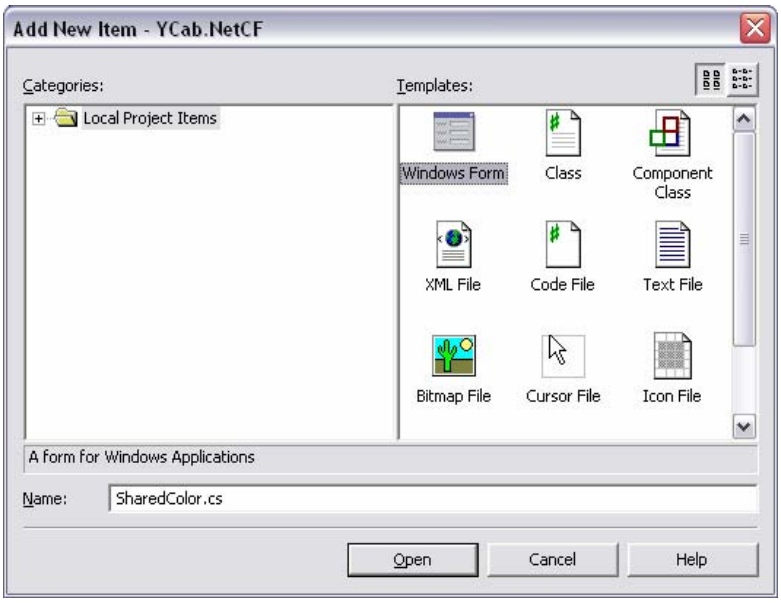


Figure 4.2 Adding a new class to the project

Make sure the icon selected in the Templates panel is that of "Windows Form". Now VS will display an empty Windows Form. This is where the GUI components will be added. Now from the toolbox panel drag a "Panel" and drop it on the center of the form. Also drag and drop a button below the panel on the form. Label the button "Next" by going to the properties menu. Now the form should look like the following figure.

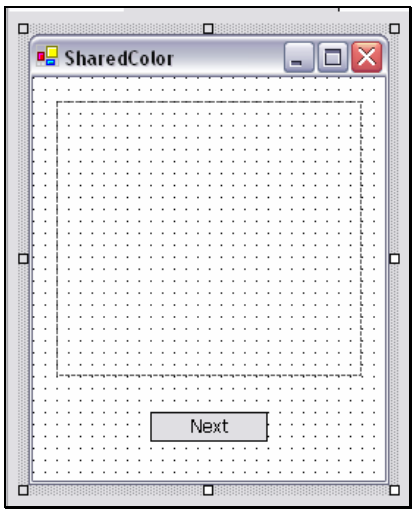


Figure 4.3 Design view of the SharedColor service

4.4 Setting the Namespace

Like in C and C++, C# also has a special reserved keyword to import additional classes. In C# that keyword is “using”. In C# reference to external namespaces can be made using the word “using”. YCab.net CF is organized by using namespaces. All the services – system level and user level – are contained in the ycService namespace. In order for the SharedColor service to work, it should also belong to the ycService namespace. In visual studio, the code for a Windows Form can be viewed by pressing the “F7” function key on the keyboard. A Windows Form in VS has two views, the designer view as shown in the figure above and the code view. By pressing F7 the code view is displayed and by pressing Shitf+F7 the design view is displayed. In the code view, set the namespace to ycService. Also at the top of the document add the ycClient namespace, as any service in YCab.net CF has to reference this namespace. After all these changes the top of the code view should look like the following.

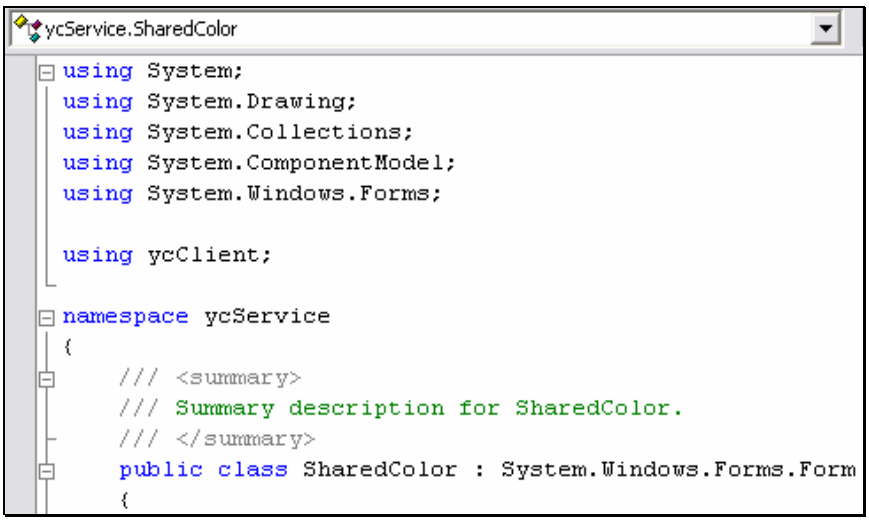


Figure 4.4 Setting proper namespaces

4.5 Setting Inheritance

All the services in the YCab.net CF API inherit either from the class Service or ThreadedService. Those services that are visible and contain GUI components inherit from the class Service and the ones that run in the background and are not visible inherit from the class ThreadedService. The SharedColor service is a visible service so it should extend the class Service. By default a Windows Form inherits from the System.Windows.Forms.Form class as shown in figure above. Now change that to inherit from class Service as shown in figure below.

```
//public class SharedColor : System.Windows.Forms.Form
public class SharedColor : Service
{
```

Figure 4.5 Setting proper inheritance

The only problem that arises because of this change is that now VS will not display the design view of the SharedColor because now SharedColor does not inherit from System.Windows.Forms.Form class, so whenever a change has to be made to the design of the form, uncomment the first line in the previous figure and comment the second line.

When done with the changes in the design of the form save the file and let the SharedColor class inherit from Service again.

4.6 Implemented SharedColor Class

At this point all the necessary setup is done. As specified in the design of SharedColor, initialize an array of colors as shown below in the constructor.

```
public class SharedColor : Service
{
    private System.Windows.Forms.Panel panel1;
    private System.Windows.Forms.Button button1;
    private Color [] colors;
    private int colorIndex;

    public SharedColor()
    {
        //
        // Required for Windows Form Designer support
        //
        InitializeComponent();

        //
        // TODO: Add any constructor code after InitializeComponent call
        //
        // init the sequence of colors
        colors = new Color[5];
        colors[0] = Color.Blue;
        colors[1] = Color.Orange;
        colors[2] = Color.Yellow;
        colors[3] = Color.Red;
        colors[4] = Color.Green;

        // init the color-index
        colorIndex = 0;
    }
```

Figure 4.6 Constructor of SharedColor class

Also declare an integer to keep track of which color to show next when the user clicks on the “Next” button.

4.7 Handling User Input

Windows programming is event driven. So until the user interacts with the SharedColor service using the keyboard or the stylus, the SharedColor service will be

idle. The moment a key is pressed on the mobile device or a stylus is touched on the screen, an event is generated. If a method is registered with that event, then it will be called or else SharedColor will ignore the event and again will become idle. As specified in the design specification of SharedColor, something needs to be done when the user clicks the "Next" button on the screen. An event handler can be associated with a GUI component by double-clicking on the control in the design view of SharedColor. When the "Next" button is double clicked in the design view, the code view is displayed as shown in figure below.

```
private void button1_Click(object sender, System.EventArgs e)
{

}
```

Figure 4.7 Adding an event handler

The method "button1_Click" is an event handler. This method will be called whenever the user clicks on the "Next" button on the SharedColor service. VS automatically registers this event handler with the button when the button is double-clicked in the design view.

When the user will click on the "Next" button, the color of the screen should be changed to the next color in sequence, so in this method a message should be sent to all the collaborators which will change everyone' s screen color to the next color in sequence. In order to accomplish this, every time the user clicks on the "Next" button, a message containing the index of the color will be sent out. The "button1_Click" method should me implemented as shown in the following figure.

```
private void button1_Click(object sender, System.EventArgs e)
{
    // convert the integer to bytes
    byte [] message = BitConverter.GetBytes(this.colorIndex);

    // send the message
    this.send(message, Protocol.ALL);

    // increment the index
    this.colorIndex++;

    // reset the index if needed
    if(this.colorIndex == this.colors.Length)
    {
        this.colorIndex = 0;
    }
}
```

Figure 4.8 Implementation of the event handler for SharedColor service

As seen in the previous figure, the method "send" is used to send the message. This method is inherited from the base class "Service". Also the second parameter of the method is "Protocol.ALL". The class Protocol is implemented under the namespace "ycClient", which is included in the SharedColor class at the beginning. Protocol class contains all the useful constants. The second argument is the recipient of the message, but as we want all the collaborators to receive this message, Protocol.ALL is used as the recipient.

4.8 Processing Incoming Messages

All the classes that inherit from the class "Service" should override a method with the signature - processMessage (byte [] msg, string sender). This method will be called by the API whenever there is a message sent by any of the other collaborators. In SharedColor the parameter msg, will contains the index of the color to be displayed on the screen. The processMessage method should be implemented as shown in the following figure.

```
public override void processMessage(byte[] msg, string sender)
{
    // get the int out of the message
    int index = BitConverter.ToInt32(msg,0);

    // set the screen color
    this.panel1.BackColor = this.colors[index];

    // point to the next color in sequence
    this.colorIndex = index+1;

    // reset the index if needed
    if(this.colorIndex == this.colors.Length)
    {
        this.colorIndex = 0;
    }
}
```

Figure 4.9 Implementation of processMessage

Now the SharedColor service is completely ready for use. The last thing that needs to be done is adding SharedColor to the application. Even though, SharedColor service is implemented under the ycService namespace, it does not automatically become part of the application. It needs to be added to the Client Frame of the YCab.net CF application. Under the namespace YCab_NetCF, there is a class named YCab_Net_CF. This class contains the “Main” method which is the method that gets executed upon application startup. In this class there is a method names “Start”. This is the method that adds all the user-level services to the client frame. The system level services are automatically added by the client frame. Add the SharedColor service to the client frame along with other services. After the SharedColor service is added, the Start method should look like the following figure.

```
public static void Start(ConnectionInfo info, Form f)
{
    // init the client frame
    ClientFrame frame = new ClientFrame(info.ip, info.port, info.userid, info.mode);

    // add the text chat service
    frame.addService(new Chat(), "Text Chat", true);

    // add the white board service
    frame.addService(new WhiteBoard(), "White Board", true);

    // add the shared files service
    frame.addService(new SharedFiles(), "Shared Files", true);

    // add the shared image service
    frame.addService(new ImageViewer(), "Shared Image", true);

    // add the audio chat service
    frame.addService(new Audio(), "Audio chat", true);

    // add the shared color service
    frame.addService(new SharedColor(), "Shared Color", true);

    // start the application
    frame.start(f);
}
```

Figure 4.10 Adding the new service to the Client Frame

When adding a service to the client frame, the first parameter is the actual object that is instantiated from the class, the second argument is the text that appears in the main menu for that service and the third parameter is the boolean value specifying if the service should be visible or run in the background.

4.9 Compiling and Deploying

Now the application is ready for compilation. In VS under the Build menu, the option “Build Solution” will compile the project. There should not be any errors if the steps described above are strictly followed. Now the application is ready to be deployed to the Pocket PC device. To deploy the application from the Build menu, the “Deploy Solution” option will install the application on the Pocket PC under the “Program Files” folder.

4.10 Output

Once the application is run, the SharedColor service will be automatically added to the main menu of the application as shown in the following figure.

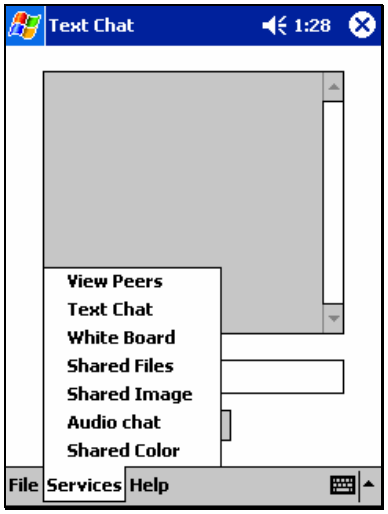


Figure 4.11 SharedColor automatically added to the main menu

Once the Shared Color service is selected, the user can rotate the color of the screen by clicking next. The first color in the predefined sequence is blue. Now if the user clicks on “Next” button, the next color in sequence is orange. This scenario is shown in the following two figures.

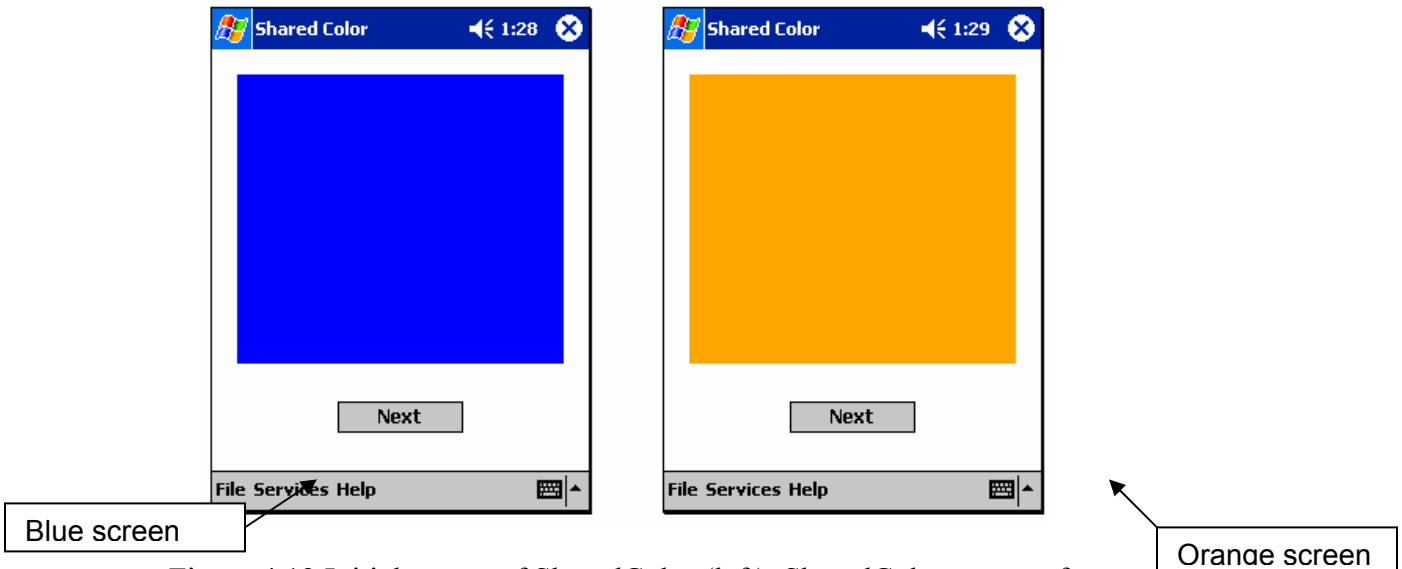


Figure 4.12 Initial screen of SharedColor (left), SharedColor screen after one event

CHAPTER 5
CONCLUSION AND FUTURE WORK

YCab.net CF is a descendant of the YCab.net, which in turn is a descendant of YCab Java. The YCab project in general has improved in terms of its use on mobile devices. YCab Java was developed for desktops and laptop computers that supported Sun Microsystems's Java Virtual Machine. Most of the small mobile devices today i.e., handheld PCs and Smart phones do not have a Java Virtual Machine, so YCab Java can not be used on such devices. Then YCab.net was developed which uses Microsoft's .NET framework. At the time YCab.net was developed, Microsoft had plans for releasing a sub-set of the .NET framework for handheld devices. YCab.net laid the foundation for YCab.net CF which is discussed in this thesis, but it could not be used to develop applications that would run on handheld devices. YCab.net CF is currently supported on handheld devices running the Microsoft PocketPC operating system. With few changes in the GUI and almost no changes in the any other part of the YCab.net CF architecture, YCab.net CF can be used to create applications for cellular phones running the Microsoft SmartPhone 2003 operating system. Thus YCab.net CF has fulfilled the long term vision of creating a true collaboration framework that can be used to create applications that would run on mobile devices such as handheld PCs and cellular phones.

Improvements in the YCab.net CF API

This section briefly explains the changes that could possible improve the overall performance of YCab.net CF API. Due to the lack of time, these changes have been made yet.

5.1 YCab.net CF Server

All the previous versions of the YCab project were developed for ad-hoc mobile collaboration so the collaboration space could be created in ad-hoc network environment only, where there the internet is not available. Also in ad-hoc mode, the collaborators are required to be within a radius of certain distance. This is because of the limitation inherited by the application from the wireless card installed on the mobile device. The version discussed in this thesis i.e., YCab.net CF has enhanced the framework by introducing a new feature that allows remote collaboration via the internet. All the collaborators connect to a central server and the virtual collaboration space is created. The Java server discussed earlier in this thesis can be improved in terms of performance and robustness. Also the server could store userid and their passwords and thus an authentication layer can be added to the current architecture.

5.2 Security

Also in the current version of YCab.net CF, all the messages are sent in form of byte arrays. The arrays contain the header information and then the actual message that is being sent. It would be difficult, but not impossible to intercept the messages and reconstruct the information that is being sent. A security layer can be added which would fit between the communication management layer and the data transfer layer.

5.3 New Services

The true value of YCab.net CF would always increase by adding new built-in services. A GPS service could be added which would display the true position of all the collaborators on a map. In this version of YCab.net CF, the audio chat service allows the collaborators to send and receive voice messages to and from each other. This service can be improved by allowing the collaborators to send and receive these messages in real-

time as if they are talking to each other using a telephone. A new service could be added which will allow the collaborators to see each other in real-time using a video camera and real-time video streaming.

LIST OF REFERENCES

[1] Archer, Tom. *Inside C#,* 2nd Edition. Microsoft Press, Redmond, WA, 2002.

[2] Buszko, D. *A Lightweight Collaborative API for Ad-hoc Mobile Computing*. Thesis for Master of Science, University of Florida, 2000.

[3] Buszko, D., Lee, W., and Helal, A. "Decentralized Ad-Hoc Groupware API and Framework for Mobile Collaboration." In *Proceedings of the ACM International Conference on Supporting Group Work,* Boulder, Colorado, USA, September 2001, pp. 5-14.

[4] Chen, Ing-Ray (2003). *Multihop Ad Hoc Messaging Using Microsoft.NET Compact Framework and Pocket PCs*. Available from URL: http://people.cs.vt.edu/~irchen/microsoft-grant/Website_HTML_Files/index.html. Site last visited April 2004.

[5] Grimes, R. *Developing Applications with Visual Studio .NET*. Addison Wesley Professional, Berkeley, CA, 2002.

[6] Gunnerson, E. *A Programmer's Introduction to C#,* 2nd Edition. Apress, Berkeley, CA, 2001.

[7] Lam, Patrick (2001), WinChat For .NET. Available from URL: *http://www.csharphelp.com/archives/archive159.html*. Site last visited April 2004.

[8] Lee, W. D. *Decentralized Ad-hoc Groupware API and Framework for Mobile Collaboration*. Thesis for Master of Science, University of Florida, 2000.

[9] Microsoft Corporation. *Microsoft C# Language Specifications.* Microsoft Press, Redmond, WA, 2001.

[10] Microsoft Corporation (1999). *Video and Audio Conferencing.* Available from URL: http://www.microsoft.com/netmeeting/. Site last visited April 2004.

[11] Platt, D. S. *Introducing Microsoft .NET*. Microsoft Press, Redmond, WA, 2001.

[12] Procopico, Michael. YCAB.NET: Decentralized Collaboration Groupware For Mobile Devices Using The Microsoft .NET. Thesis for Master of Science, University of Florida, 2002.

[13] Rubin, Erik and Yates, Ronnie. *Microsoft .NET Compact Framework Kickstart*. Sams Publishing, Indianapolis, Indiana, 2003.

[14] Weisman, C. J. *The Essential Guide to RF and Wireless,* 2nd Edition. Prentice Hall, Upper Saddle River, NJ, 2002.

[15] Wigley, S. Wheelwright, R. Burbidge.*Microsoft® .NET Compact Framework (Core Reference)*. Microsoft Press, Redmond Washington, 2002.

BIOGRAPHICAL SKETCH

Mihir P. Patel was born and raised in Gujarat, India. After finishing his high school, he came to the United States in June 1998. He graduated with Associate of Arts degree from Palm Beach Community College in 2000. He then attended the University of Florida and received his Bachelor of Science degree in computer engineering in May 2004. He is a graduate student at the University of Florida in the Department of Computer and Information Science and Engineering. At present he works part time as a software developer for the Institute of Food and Agricultural Sciences at the University of Florida. His interests include mobile computing and game development for mobile devices. For more information he can be contacted via email at immihir@hotmail.com.