Electrical and Computer Engineering Department

Marko Pascan

Technical Report ECE.P54.2006.5
August 2006.

University of New Hampshire
Electrical and Computer Engineering Department
CATLab

AroundMe Project using Project54 Platform

By Pascan Marko
August 2006.

Acknowledgments

First of all, I would like to thank Prof. Andrew Kun for giving me the opportunity to be part of the Project54 team and to grasp new knowledge and experiences. His guidance and advice were of most importance to my project.

Also, I would like to thank the Project 54 team, especially Christopher Gaudreau for his help every time I had a question or problem with the software or hardware and other members of Project54 who helped me with my work.

Finally I would like to thank Prof. Vladimir Katic for helping me realize the research opportunity presented to me this summer at UNH.

Table of contents

1. Introduction

There are countless situations where the information about people who are near you (around you) can be very useful. Here is one scenario: there was a major traffic accident on a highway and multiple policemen have arrived at the scene. They all have PDAs and can start communicating to each other via a wireless network to share data about the accident and victims, or send messages. Another scenario is a business meeting. It is very useful to know something about the people seated with you if you do not already know them. This is a very interesting problem and once a system with basic functionalities is developed one can easily upgrade it and add some other functionalities.

Of course, many people have already worked on similar problems. There are a handful of systems that already have been developed. The aim of the AroundMe project was to use some of these concepts to build an application that will work on the Project54 framework.

The aim of this project was to build an application for a PDA computer that would enable the user to know which other PDA computers are around him, and which PDA computers are around other PDA computers. Once users have this information they can start communicating with other PDA computers either directly or indirectly (routing). This is a brief description of a basic application that can be further developed.

The first step was to find documents about similar systems that have been developed and study them. These papers gave me insight into the technology and concepts that have been used, as well as difficulties that people encountered while working on these projects.

The second step was to find out which parts of the Project54 framework could be used and how to develop such a system. Some of the tools and libraries that I used were not from the Project54 framework.

The third step was to develop a test application that uses some concepts and parts of the Project54 framework.

And finally the last step was to develop a prototype application, testing and documenting it.

2. Related work

2.1 The NearMe Project

NearMe [1] was developed by Microsoft Research in Redmond, USA. It consists of a server, algorithms and application programming interfaces (APIs) for clients equipped with 802.11 wireless networking (Wi-Fi) to compute lists of people and things that are physically nearby. NearMe compares clients' lists of Wi-Fi access points and signal strengths to compute the proximity of devices to one another. They have developed several clients including one for desktop computers and one for PDA computers. All of the clients work in a similar way and have several common steps:

- **Register with the proximity server** – after registering, the client receives a GUID (globally unique identifier) which is used by the server to identify which data to associate with which user.
- **Report Wi-Fi signature** – once registered with the server, a client can report to the server access points that it can see and measure signal strengths. The set of MAC (Media Access Control) addresses and signal strengths is the Wi-Fi signature.
- **Query the nearby people and places** – first the user selects a type to query for. The server returns two lists of nearby instances of requested type: short range proximity (instances that have at least one access point in common with the querying client) and long range proximity (instances that can be reached by hopping through access points with overlapping coverage, sorted by the number of hops required).

The NearMe server is an SQL database that maintains tables of active users, static resources (like printers and conference rooms) and their associated Wi-Fi signatures. It also maintains metric and topological data about the physical layout of access points derived from Wi-Fi signatures.

2.2 Relate project

Relate [2] was developed by the Computing Department of Lancaster University. It uses location information to enhance communication between mobile devices. It requires instrumentation of environment. This system allows devices to measure their spatial relations in peer-to-peer fashion using extensions for mobile devices. These extensions make information on spatial relations available to support interaction with nearby users, devices and resources. This system consists of several elements:

- **USB dongle -** are sensor nodes that can be attached to mobile computing devices such as laptops and PDAs via USB. Dongles can measure distance and angular bearing between one another in true peer-to-peer fashion. These devices use ultrasound and they are custom built (casing, chip and three ultrasound sensors).

- **Relate Spatial Engine -** is software that runs on the device to which the dongle is attached. It is responsible for computing and updating high-level models of spatial relations.
- **Relate toolkit -** is a set of APIs for building user interfaces that make use of spatial relations.

A Relate dongle has three ultrasound sensors that cover space in front and on the sides of the dongle. It emits sense and analyzes signals as well as collects data which is uploaded to the Relate Spatial Engine when needed. This engine is software that runs on the mobile host. It interfaces with the dongle to receive data from it. Data is needed to compute and maintain a dynamic spatial model as real time representations of arrangements of mobile devices. It also establishes a peer-to-peer overlay network. The spatial arrangement of devices is modeled in a graph structure. The Relate toolkit is set of APIs written in the JAVA programming language. It enables developers to build graphical user interfaces that make use of spatial relations. Creators of this system built several applications using Relate, for example:

- Spatial awareness support for meeting
- Spatial file transfer

2.3 Agent Network for Bluetooth Devices (ANBD)

ANBD [3] was developed at The University of Palermo. They developed a mobile agent for personal mobile devices that uses these devices as adaptive human environment interfaces to supply people with ad-hoc information and high-level services. It is called Agent Network for Bluetooth Devices. The ANBD system operates with a hierarchical framework of service providing nodes, dynamically composed and managed by mobile agents. One of the key characteristics of this system lies in its ability to dynamically adapt itself to environmental changes. This network consists of devices that belong to two groups:

- **Mobile devices** - PDAs, mobile phones, laptops… They must have a Bluetooth connection and J2ME (Java Micro Edition) execution environment.
- **Fixed devices** – such as PCs exploited as agent servers or databases, Bluetooth access points.

Environment in which ANBD is used is divided into logical areas. There are two main area types:

- **Service areas** – smallest entity within the ANDB framework. It consists of networked Bluetooth base stations.
- **Resource areas** – consists of a federation of one or more lower-level areas that the control server manages. The highest RA is the root area that contains all of lower level areas.

There are several applications that have been developed using this system (document request, exam schedule, exam enrollment…). Once a mobile device is in range of this network and is discovered, the Bluetooth service pushes MIDlet onto them. After installation users can use services from mobile devices.

2.4 Project Aura

Aura [4] was imagined as a "distraction-free pervasive computing" system. Aura was developed at Carnegie-Mellon University. As one of the most precious resources is human attention, Aura aims to minimize distractions to user attention, creating an environment that adapts to user's context and needs. Human attention refers to a user's ability to attend to his primary tasks, ignoring system-generated distractions (poor performance, failures). Project aura is specifically intended for pervasive computing because user's attention is very scattered in those systems. Components which Aura uses already exist. But, the whole system is far more complex than the sum of its parts. Software implements concepts that amplify the capabilities of a resource-limited mobile client and thus improve user's experiences. Aura also defines wireless bandwidth advisor. That is software that can use reasonable estimates of future available bandwidth to make informed decisions such as server selection. Also, Aura has a people location service which is based on signal strength and access point information from a IEEE 802.11 wireless network. Aura can use two algorithms for location sensing:

- Pattern-matching algorithm
- Triangulation-based remapped interpolated algorithm

Two of Aura's most important capabilities are supporting user mobility and shielding users from variations in resource availability. This is called capturing user intent. Several context -aware applications were built using Aura infrastructure.

3. Project Preview

3.1 Project54 Platform

Project AroundMe is based on the Project54 platform. It is a separate application (component – AroundMe.dll) that is connected to the rest of the application through the interface. It uses the Application Manager and the Proxy application extensively for messaging protocols that are implemented in the AroundMe software. The message format used in the application for sending and receiving data from other applications is version 2 messaging [5] (UDP/IP message format). The format of a version 2 compatible message in a UDP/IP packet payload is as follows:

<P54V2>source; destination; identifier; message_body

As the AroundMe application needs to send more data than simple command messages, additional data and commands were embedded in the message body using tokens extensively. Several new message formats were introduced and the different form version 2 messages are only in the message body which is used for carrying additional data. The graphical user interface is based on the Project54 GUI library. As this GUI library is poor in widgets that a developer can use, some of the windows are not so user-friendly.

3.2 External libraries

For access point scanning functionality the AroundMe application is using functions from a library written by John Krumm (one of the creators of the NearMe project mentioned above). Functions from this library are exported in GetRSSILib.dll, which the application uses for extracting functions for access point scanning.

3.3 AroundMe functionalities

Main functionalities of AroundMe are:

- Setup destination PDA to which you want to send a message.
- Setup the message that is going to be sent
- Ping PDA computers in surrounding area and show data about "who can see who" on the screen
- Manually route messages. In this version of routing only one hop is possible.
- Automatically route messages. In this version multiple hops are possible.
- Setup PDA's application name from which you want to receive access point scan data
- Access point scan

4. Project Status Overview

4.1 Introduction

For a functional system at least two PDA computers are needed. Some of the functionalities implemented in AroundMe can be used only with one PDA computer (access point scan). But, main functionalities are really visible only in a system with multiple PDAs involved. Most of the developing and testing has been done in a system consisting of three PDA computers.

4.2 Ad-Hoc Network

The full name of ad-hoc networks is mobile ad–hoc network (MANET). It is a self-configuring network of mobile routers (or devices) connected by wireless links – the union that forms an arbitrary topology. Members of an ad-hoc network (devices, routers…) are free to move randomly and organize themselves arbitrarily, thus the networks wireless topology can change rapidly and unpredictably.

As AroundMe application's main functionality is to send and receive messages and data from other PDA devices they have to be in some kind of network. We picked an ad-hoc network. One was created between PDAs running the AroundMe software. Every PDA is using a specific IP address. After creating an ad-hoc network between PDA devices they can communicate with each other.

4.3 Test application

In order to test how the Proxy application is acting in the case of sending messages to other PDA devices, I built a simple test application. This application is sending a simple message to other PDA devices. In order to send a message, the sender must know the name of the destination application. If the application manager receives a message that is for an application that is not installed on the source PDA it will forward the message to the Proxy application. The Proxy application must know the IP address of the destination application (PDA). That information should be written in a special textual file – registrysettings.txt. Only in this case will the Proxy application will open the socket towards the destination IP address and send messages to the Proxy application on the other side. After receiving messages from the remote application, the Proxy application on the receiving side will send the message to the application manager because from the settings file it knows that the destination application is installed on the receiving device. The Application manager will interpret message in certain ways (for example it will write a message on the screen). This is a simple description of this test application. Here is a piece of code that is sending a message to an application that is not installed on the source device:

```
// Query button click
if (hact == hbcQuery)
{
      int state;
      getBCState(hwAroundMe, hbcQuery, &state);
      if(!state)
      {
            wchar_t wRegistryPDA[16];
            int ibuffer = 16;
            bool bRegistryRead = getRegStringValue(L"AroundMe",
            L"PDASettings", L"AppName", wRegistryPDA, &ibuffer);
            if(!bRegistryRead)
            {     setTAText(hwAroundMe,htaData,L"Please, setup remote
                  PDA application");
                  bUpdateAroundMeGUI = true;
            }
            else
            {
                  Message(self,wRegistryPDA,self,L"TEST_MESSAGE");
            }
      }
}
```

Fig. 1: Sending of “TEST_MESSAGE” message in message body part. Destination application name is in wRegistryPDA variable which has been filled from registry.

Next code example shows how the destination application interprets the message after receiving it:

```
if(wcscmp(message, L"TEST_MESSAGE") == 0)
{
      addMessage(L"How are you?");
}
```

Fig 2: In this case the destination application writes a message on the screen after receiving the message

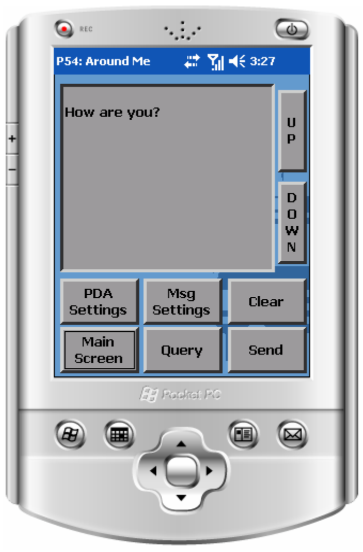


Fig 3: Receiving PDA with message shown on the screen

4.4 User manual

This is a short user manual that explains how to use the AroundMe application.

4.4.1 Main screen

After running the Project54 application manager for PDA devices, the main screen of the Project54 application appears on the PDA's screen. By clicking on the "**AroundMe**" button the user starts the AroundMe application. Clicking on the "**Exit**" button will terminate the Project54 application manager.

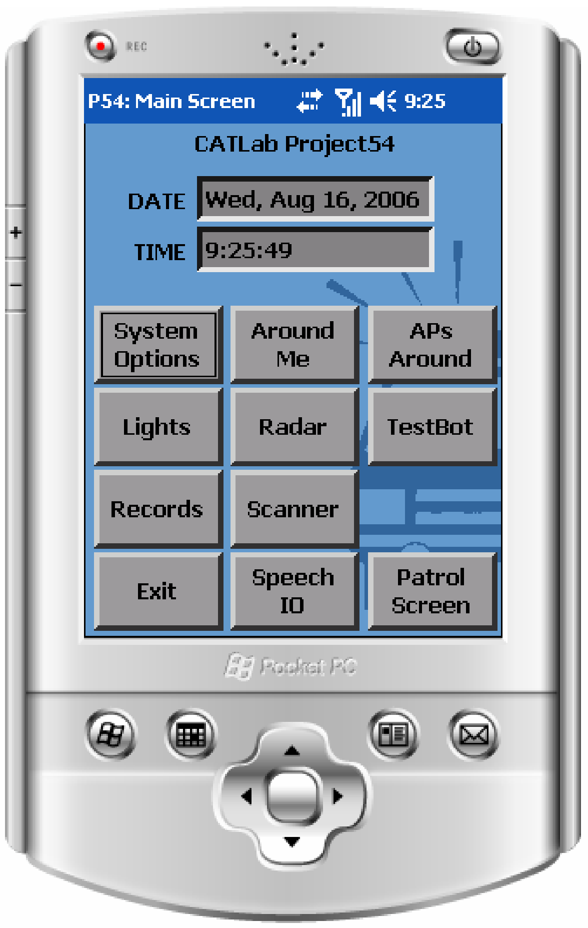


Fig 4: Main screen of Project54 application for PDA devices

4.4.2 AroundMe Main Window

AroundMe application main screen consists of following GUI elements:

- **Main Screen** button – opens application manager's main screen
- **Ping** button – calls ping function that recursively pings all remote AroundMe applications known to PDA which is pinging.
- **Send** button – sends chosen (or created) message to chosen remote AroundMe application.
- **Sending Settings** button – opens "**Sending settings**" window, where user can set up destination remote AroundMe application

and remote AroundMe application that will be used as router in case of manually routing of messages.

- **Message settings** button – opens "**Message Settings**" window, where user can create new or choose existing message that will be sent to remote AroundMe application.
- **Clear** button – clears the text area for showing data
- **Scan settings** button – opens "**AP Scan Settings**" window, where user can choose from which remote AroundMe application to ask for data about access points around the PDA which carries that application. By access point data I mean MAC address and signal strength of every access point that wireless adapter of the PDA chosen in this settings can "see".
- **AP Scan** button -triggers access points scan function
- **Automatic Routing** button – when this button is pushed down routing of messages is performed automatically.
- **UP** button – scrolls data text area up
- **DOWN** button – scrolls data text area down
- Text area for displaying data

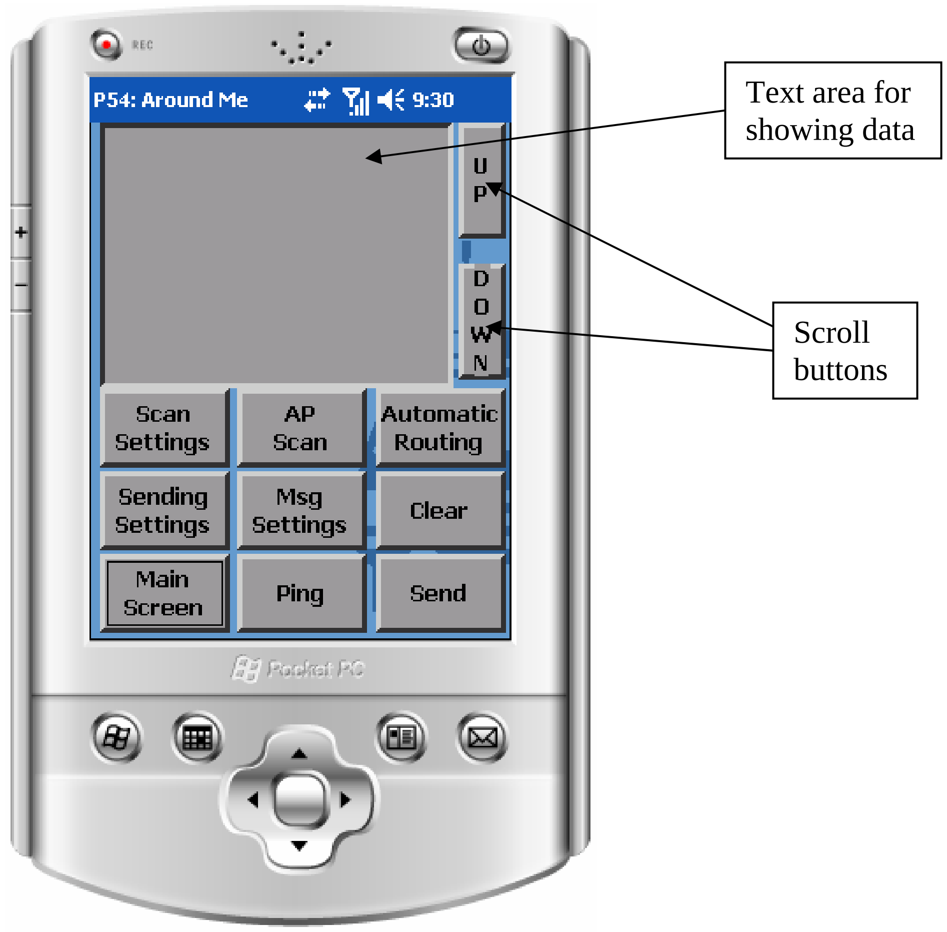


Fig 5: Main window of AroundMe application

4.4.3 Sending Settings Window

Sending Settings window is consisted of following GUI elements:

- **Back** button – opens AroundMe main screen
- **Next Via** and **Prev Via** buttons - sets next/previous name of the remote AroundMe application from a list of remote AroundMe applications known to this application into "Via" edit box. That application will act as a router for manually routed messages. Names and parameters of remote AroundMe applications, with which applications can communicate, are stored in the **registrysettings.txt** and are read during the initialization of the AroundMe application and stored in the list for further usage. If a user sets the "Direct" message into the "Via" edit box, the message will be sent directly to the destination.
- **Next PDA** and **Prev PDA** buttons – sets next/previous name of remote AroundMe application from a list of remote AroundMe applications known to this application into "Send to" edit box. That application is the destination application for the message that is going to be sent from the PDA device on which these settings are executed. Names and parameters of remote AroundMe applicationss, with which applications can communicate, are stored in **registrysettings.txt** and are read during the initialization of the AroundMe application and stored in the list for further usage.
- **Save** button – saves values from edit boxes into registry.

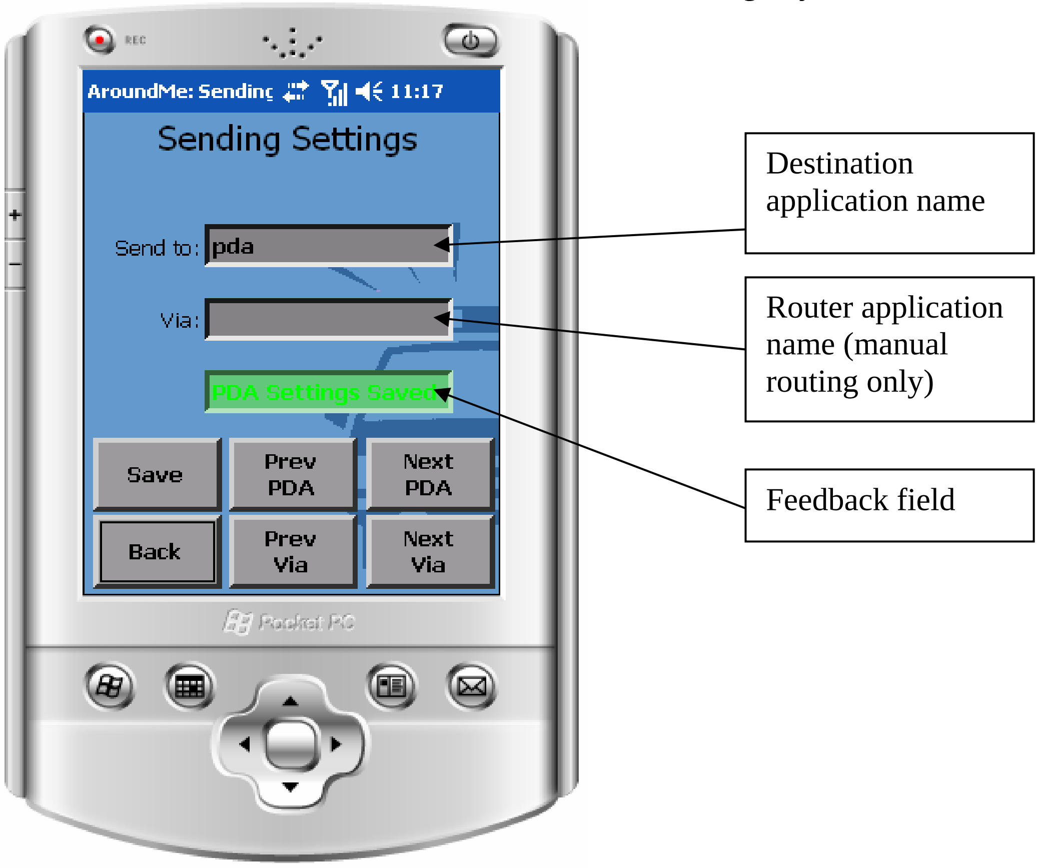


Fig 6: Sending Settings window

4.4.4 Message Settings Window

The Message settings window consists of the following GUI elements:

- **Back** button - opens AroundMe main screen.
- **Save** button – saves current message (message displayed in the big edit box) to the registry and calls function that saves all messages from message list to registry. Message list holds messages that user can chose and save as message that is going to be sent to destination. Message saved in registry will be sent when user clicks “Send” button on AroundMe application main screen.
- **New** button – clears the edit box for displaying messages and creates new empty message. After this user can type in new message.
- **Delete** button – deletes current message from message list. If there are no messages left in message list (the last message from the list has been deleted) default message will be displayed in the big edit box.
- **Prev Msg / Next Msg** buttons – displays previous/next message from message list in big edit box

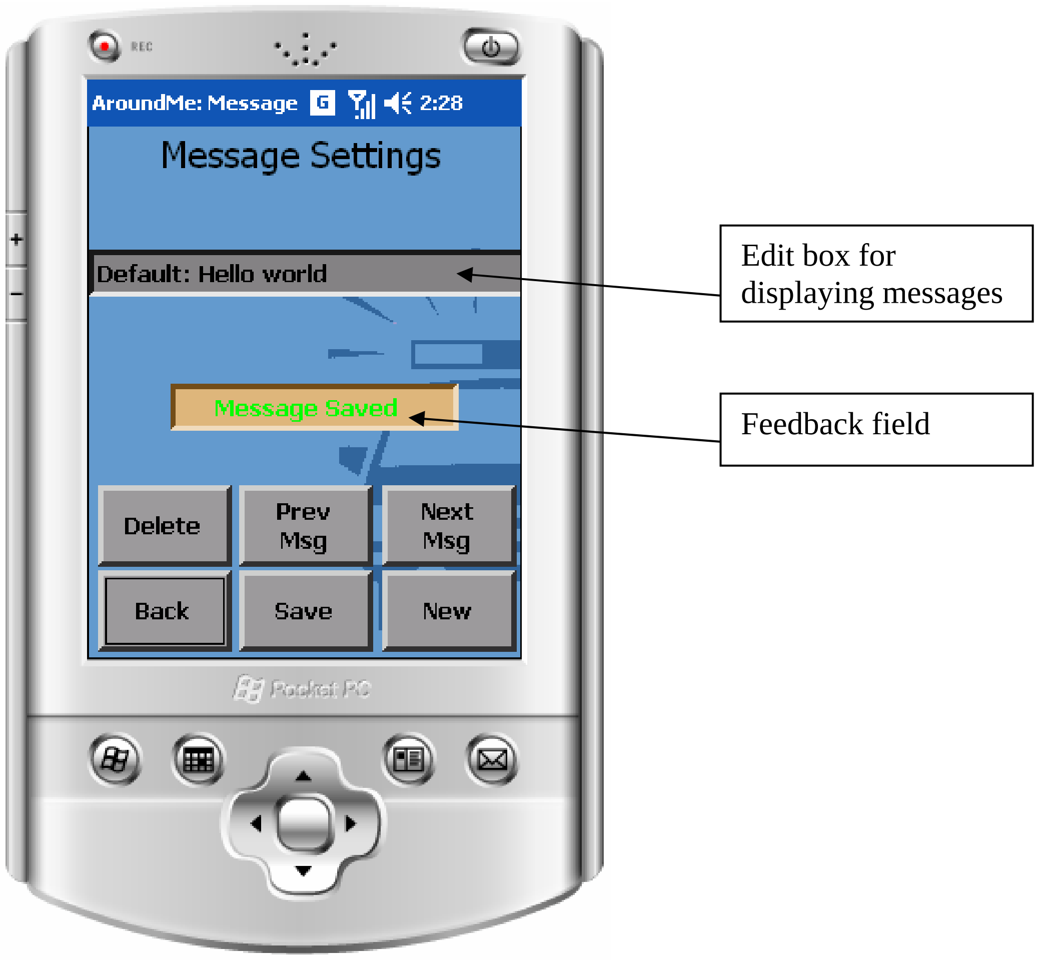
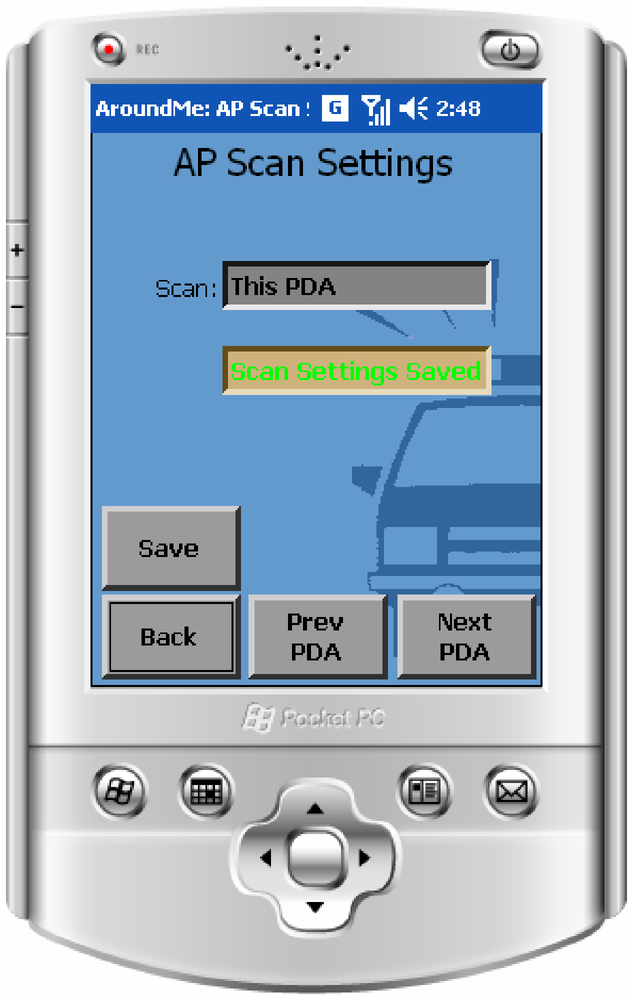


Fig 7: Message settings window

4.4.5 Scan Settings Window

Scan settings window consists of following GUI elements:

- **Back** button - opens AroundMe main screen.
- **Prev/Next PDA** buttons – puts previous/next AroundMe application name into the "Scan" edit box. Names and IP addresses of remote AroundMe applications that this AroundMe application "sees" and knows about are stored in the list that is populated during the initialization of the AroundMe software. These two buttons are in fact a way for the user to navigate through this list. There is one extra element of the list that represents the name of the application of the PDA where these settings are being executed. That means that the user can ask this application to scan its surroundings for access points. The PDA device on which the user is changing settings is labeled with the name "This PDA".
- **Save** button – saves the current AroundMe application's name from the "Scan" edit box to the registry. The Feedback message is displayed after clicking this button to indicate that parameters have been saved.


AroundMe: AP Scan : 2:48

AP Scan Settings

Scan: This PDA

Scan Settings Saved

Fig 8: Access point scan settings window

5 Main functionalities

5.1 Ping

Ping is a computer network tool used to test whether a particular host is reachable across an IP network. Ping works by sending echo-request packets to the target and listens to echo-response replies. Using interval timing and response rate, ping estimates the round-trip time and packet loss rate between hosts.

Ping functionalities implemented in the AroundMe application work in a similar way. As every AroundMe application has a list of remote AroundMe applications, the ping function sends ping messages (that is in fact an IP/UDP packet) to all applications from that list and waits for an answer for a certain period of time. If it gets an answer than the host is reachable (PDA device is in range, AroundMe application is on). If it doesn't get an answer, after that predefined period of time than the host is unreachable (PDA device is not in range, AroundMe application is off). After getting a ping message, the remote AroundMe application than pings all applications from its remote AroundMe application list (except from sender) and so on. This means that the ping function is in fact a recursive function. It eventually stops because an application never pings back applications from which a ping message came. For example, in the branch on the left (one that consists of PDA1, PDA2, PDA3 and PDA4 in this exact order), PDA3 will not ping PDA1 or PDA2, but will ping PDA4. When a ping message reaches some application that has no application to ping that means that message reached a termination node of this tree. A Termination node triggers a reply message which is sent back up the tree. The next picture shows how a ping message is distributed in an ad-hoc network consisting of 4 PDA devices:

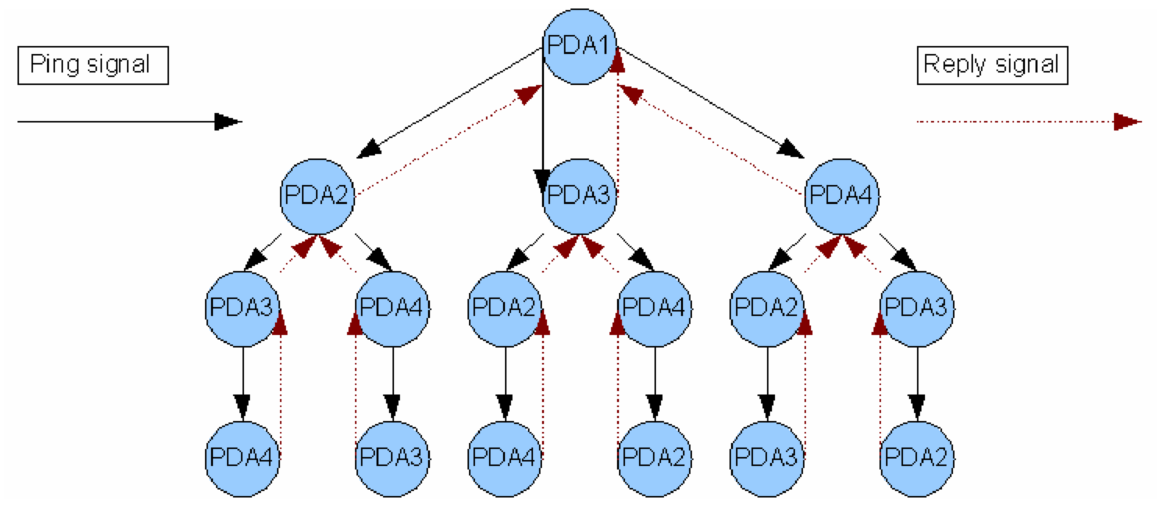


Fig 9: Diagram that shows how ping and reply signals are distributed

The reply message memorizes which PDAs it passed as it passes PDA devices on its way back up the tree towards the ping initiator. In this way, when a reply message finally reaches the ping initiator, it will have a list of PDA device names (remote AroundMe application names) that are reachable by ping message. For example, from the above picture, if a ping signal starts from PDA1 and goes through PDA2, PDA3 and reaches PDA4, which is in this example a termination node for this branch, a reply message will have the following list of PDA devices when it reaches PDA1:

PDA2, PDA3, PDA4

This is in fact one route that can be used by any message. So, for example, if a user wants to send message to PDA4, he can do that by sending a message via PDA2 and PDA3 (using PDA2 and PDA3 as routers).

5.1.1 Ping Message Format

An application that is pinging a remote application is sending message which has a special format of message body as part of the P54V2 message:

PING%FROM_APPLICATION_LIST

A ping message is identified with a PING string at the beginning of the message body. After that is a separator character '%'. "From application list" is a list of application name strings appended after the beginning of the message. Each name is again separated from each other, but with a different separator – '|'. So, ping message in fact looks like this:

PING%APPLICATION_NAME|APPLICATION_NAME|…

As this message travels down the tree from Figure 9 each PDA device (AroundMe application) adds its name to the end of this message.

5.1.2 Ping Activity Diagram

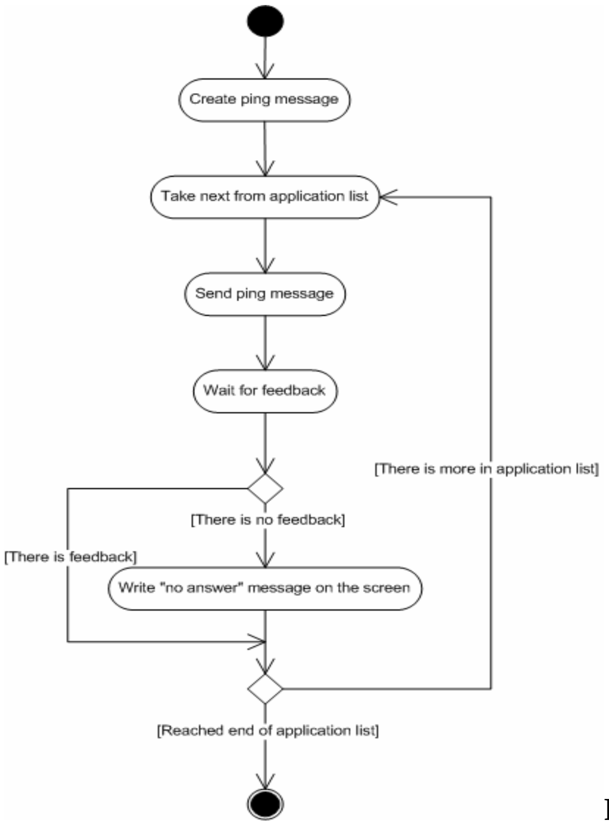

Create ping message
Take next from application list
Send ping message
Wait for feedback
[There is more in application list]
[There is no feedback]
[There is feedback]
Write "no answer" message on the screen
[Reached end of application list]

Fig 10: Ping initiator diagram

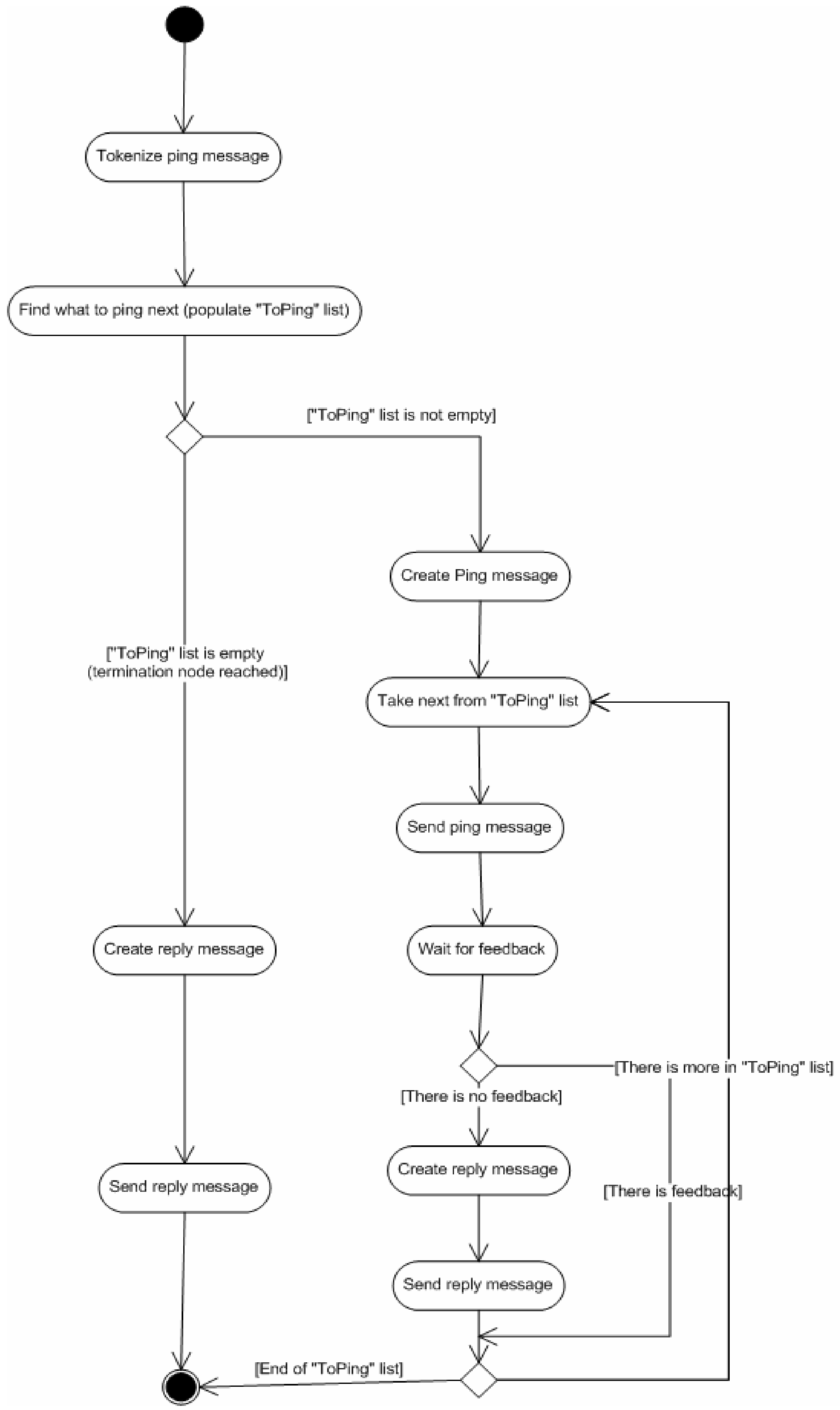
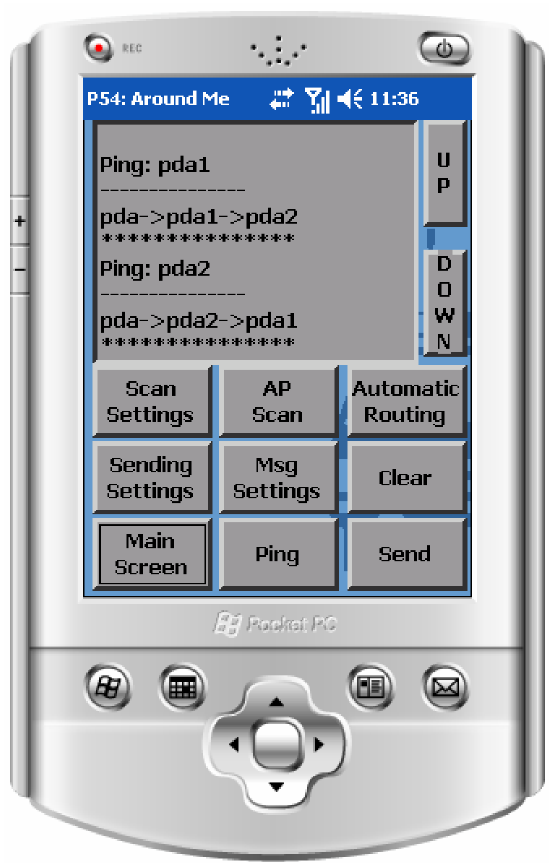


Fig 11: Activity diagram for one of PDA devices in the tree from Fig 9.

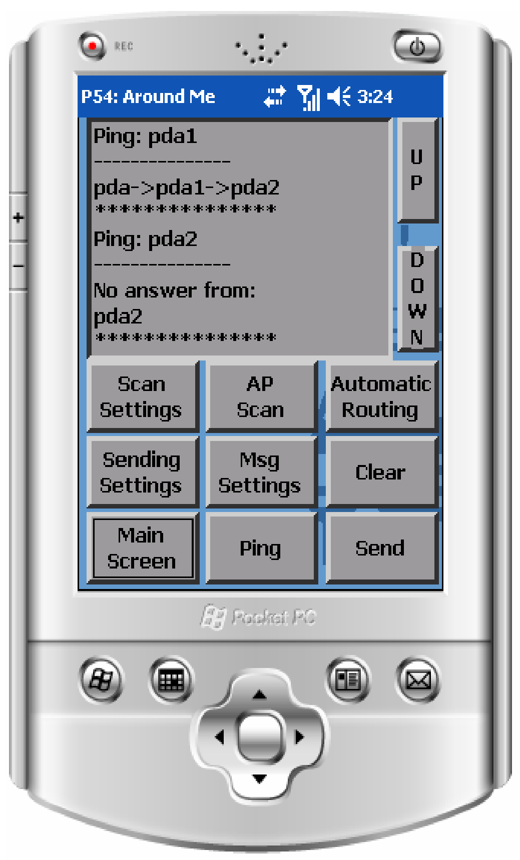
5.1.3 Ping data displaying

After pinging, results of the ping will appear on the screen. The data shows “who sees who“. The following example is with 3 PDA devices:



```
Ping: pda1
---------------
pda->pda1->pda2
****************
Ping: pda2
---------------
pda->pda2->pda1
****************
```

Fig 12: Ping data displayed.



```
Ping: pda1
---------------
pda->pda1->pda2
****************
Ping: pda2
---------------
No answer from:
pda2
****************
```

Fig 13: Example where there is no answer From PDA2, but PDA1 can see it.

5.1.4 Reply Message Format

The reply message is created and sent in two cases during the ping operation:

- When a ping message reaches a termination node
- When there is no feedback from the pinged remote AroundMe application after the waiting time has elapsed

In both cases the message body of the reply message is the same:

REPLY%SEND_TO_APPLICATIONS%FROM_APPLICATIONS

REPLY is the identifier of the message and it is used to recognize the message. The second part is the list is separated application name strings which represent the route through which the reply message must pass to

reach the ping initiator. At the start this part is in fact a copied list of applications from the ping message. As the reply message travels back to the ping initiator, every application erases its name from this string separated list. At the end this list is empty. The separator character is ‘|’.
The third part of the reply message is a list of applications through which the reply message passed. As it travels back to the ping initiator every application on the way adds its name to this part of the message. This is also a string separated list with the same separator character (‘|’). At the end this list consists of all the application names through which the reply message has passed and it is in fact the route for messaging. The full reply message format is as follows:

REPLY%APP_NAME|APP_NAME|…%APP_NAME|APP_NAME…

5.2 Manual Message Routing

Manual message routing was one step in developing a fully automatic routing mechanism. It is limited to only one router between the sender and receiver of the message. Since I have been developing this application on a system consisting of three PDA devices this was the first version that I built. The ping function’s only purpose in this case is for the user to see, from ping data, if he can reach the destination that he wants. The user will decide which remote AroundMe application to use as the router and he can choose and use only one, or if the destination application is directly reachable he can send a message directly to the destination. By setting the “Direct” string into the “Via” edit box on the “Sending Settings” window he chooses to send the message directly. By setting some application name into that same edit box he chooses to use that application as a router. After the message has been sent the application waits a certain period of time for a delivery report. If the message is delivered to the destination, then that application sends a reply message (delivery report) to notify the sender that the message has been successfully delivered.

5.2.1 Message formats

Manual message routing has several types of messages and message formats.

- **Direct message sending** – to send a message directly to the destination the user has to set the string “Direct” into the “Via” edit box in the “Sending Settings” window. After setting the destination and message, by clicking the “Send” button on the main screen of the AroundMe application, the next message will be sent within the message body:

  DIRECT_MESSAGE%MESSAGE_TEXT

  This is only the format of the message body of the P54V2 message type. The first part of the message is the indicator that the application that receives it is the final destination of this message. The second part is the actual message text. The parts of the message are separated by a ‘%’ character.

- **Indirect message sending –** if the user sets the application name into the "Via" edit box on the "Sending Settings" window, that application will be used as a router. After clicking the "Send" button on the main screen of the AroundMe application the next message will be sent within the message body:
  INDIRECT_MESSAGE%DESTINATION%MESSAGE_TEXT

  This is also only in the message body of the P54V2 message type. The first part of a message is the indicator for the receiver of this message to forward it to the destination application. The second part of the message represents (final) the destination application name. The third part is the actual message text. The parts of the message are separated by a '%' character. This message is sent to the application that acts as a router and that application resends this message but with a different indicator – VIA_MESSAGE. In this way the destination application will know that message came from the application that acted only as a router.
- **Reply to direct message delivery –** after an application receives a message with the DIRECT_MESSAGE indicator at the beginning of the message body it sends a message with the indicator DIRECT_MESSAGE_REPLY. The application which receives this message knows that the message that has been sent from it has been delivered successfully to the destination.
- **Reply to indirect message delivery –** after receiving a message from an application that acted as a router, the destination application will send a reply message to the router application with the indicator VIA_MESSAGE_REPLY. The router application will resend the reply message with the indicator DIRECT_MESSAGE_REPLY to the source application.

5.3 Automatic Message Routing

As manual message routing is not flexible and generic, development of an automatic version was the next logical step. The idea was to free the user from any additional settings except for the destination application setup. The user's only task will be to setup the destination and message and to click the "Send" button on the main screen of the AroundMe application. Multiple hops are possible if this kind of routing is used. For example: we have a system consisting of 4 PDA devices (PDA1, PDA2, PDA3 and PDA4) and PDA1 wants to send a message to PDA4. The problem is that after sending a ping message to all the other PDA devices PDA1 discovers that it can not "see" PDA4 and PDA3 but can "see" PDA2. On the other hand PDA2 also can not "see" PDA4 but can "see" PDA3. And PDA3 can "see" PDA4. The message in this case will be sent first to PDA2, which is going to route it to PDA3 and PDA3 will finally send it to the destination device – PDA4. This is called multiple hopping. In manual routing the user had to decide which remote AroundMe application to use as a router after seeing ping data. In this case ping data is stored into data structures and a function later decides which route to use for message sending. In figure 9 the distribution of a ping message is shown. When the reply message initiated by one of the termination nodes of this tree reaches the ping initiator it will have data about all PDA devices that responded to the ping message. If one of the PDA's on the way does not reply to the ping message, that route will end at that point. In the system above if every PDA "sees" everybody else the routes will be:

- PDA1, PDA2, PDA3, PDA4
- PDA1, PDA2, PDA4, PDA3
- PDA1, PDA3, PDA2, PDA4
- PDA1, PDA3, PDA4, PDA2
- PDA1, PDA4, PDA2, PDA3
- PDA1, PDA4, PDA3, PDA2

Another good example of the concept will be if one of the PDA devices is not visible by other. If PDA2 can't see PDA3 routes will be:

- PDA1, PDA2
- PDA1, PDA2, PDA4, PDA3
- PDA1, PDA3
- PDA1, PDA3, PDA4, PDA2
- PDA1, PDA4, PDA2
- PDA1, PDA4, PDA3

So as you can see, a route ends at a point where there is no answer from the next PDA device in the tree. The tree node becomes a termination node if there is no answer from a lower node in the tree.

5.3.1 Collecting routes data

Routes data is collected in a message handler for the REPLY message. In order to collect routes data a user must first ping the PDA devices. Every route is stored in a container class. This is the header file of that class:

```
/**
      \file Route.h
      \brief Contains declaration of Route class that
      represents one route for message sending
      \author Pascan Marko
      \date July - August 2006
*/

#include "dlist2.h"

class Route
{
private:
      dlist* nodeList;
public:
      Route();
      dlist* getNodeList();
      void addNode(void* newNode);
};
```

Fig 14: Route class

Instances of this class are keeping the application names of one route. Route objects are then stored in a list that represents the list of all routes found during a ping action.

5.3.2 Finding Suitable Route

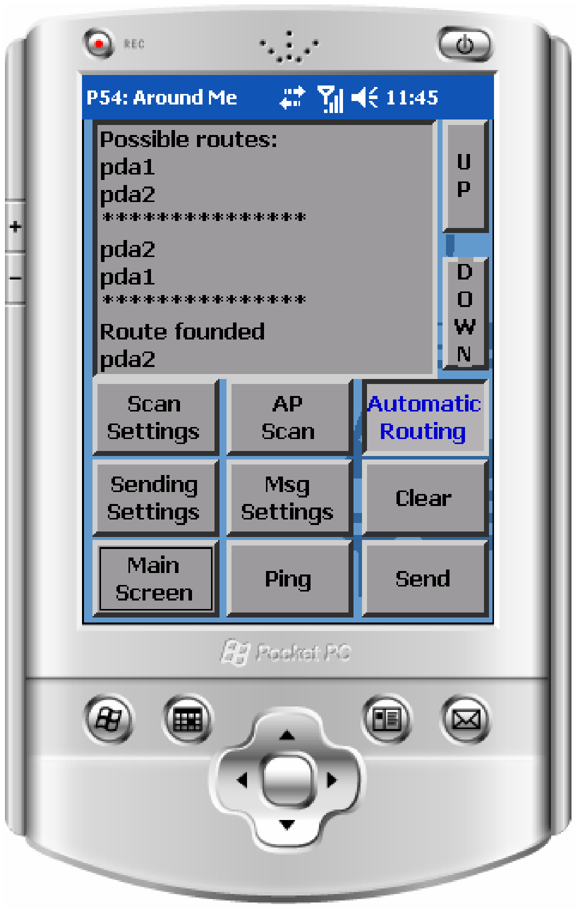
After collecting data of all possible routes, a suitable route must be found for sending messages. In other words, the best route is the shortest one. This is done in the "FindRoute" function:

```
/**
      \fn FindRoute()
      \brief Function that finds shortest route to destination pda
      (automatic routing of messages in ad-hoc network)
      \return Route for message
*/
Route* FindRoute()
{
//DEBUG////////////////////
addMessage(L"Possible routes");
routesList->home();
for(int t = 0; t < routesList->get_size(); t++)
{
      Route* rot = (Route*)routesList->get_data();
      dlist* temp = rot->getNodeList();
      temp->home();
      for(int s = 0; s < temp->get_size(); s++)
      {
            wchar_t possibleRoute[16] = L"";
            wcscat(possibleRoute,
            getFriendlyName(((AppNameType*)temp->get_data())->appname));
            addMessage(possibleRoute);
            temp->next();
      }
      addMessage(L"****************");
      routesList->next();
}
////////////////////////////
Route* retVal = NULL;
wchar_t wRegistryPDA[16];
int ibuffer = 16;
bool bRegistry = getRegStringValue(L"AroundMe", L"PDASettings",
L"AppName", wRegistryPDA, &ibuffer);  // read destination app name from
registry
if(bRegistry)
{
      routesList->home();
      dlist* currentNodeList;
      Route* currentRoute;
      for(int i = 0; i < routesList->get_size(); i++)
      {
            currentRoute = (Route*) routesList->get_data();
            currentNodeList = currentRoute->getNodeList();
            currentNodeList->home();
            // can i send message directly?
            if(wcscmp(wRegistryPDA,
            ((AppNameType*)currentNodeList->get_data())->appname) == 0)
            {
                  retVal = new Route();
                  retVal->addNode((AppNameType*)currentNodeList-
                  >get_data());
                  return retVal;
            }
            routesList->next();
      }
      // if i can't send it directly, than find shortest route
      routesList->home();

      ///////////////////////////////////////////////////////////////////
      // collect all the routes to destination and take the shortest one
      ///////////////////////////////////////////////////////////////////
      Route* route = NULL;
      dlist* routesFounded = new dlist();
      // collect all routes
      for(int j = 0; j < routesList->get_size(); j++)
      {
            route = new Route();
            currentRoute = (Route*) routesList->get_data();
            currentNodeList = currentRoute->getNodeList();
            currentNodeList->home();
            for(int k = 0; k < currentNodeList->get_size(); k++)
            {
            route->addNode((AppNameType*)currentNodeList->get_data());
            if(wcscmp(wRegistryPDA,
            ((AppNameType*)currentNodeList->get_data())->appname) == 0)
            {
                  routesFounded->add_data(route);
                  break;
            }
            currentNodeList->next();
            }
            routesList->next();
      }

      // find shortes route
      routesFounded->home();
      retVal = (Route*)routesFounded->get_data();
      for(int n = 0; n < routesFounded->get_size(); n++)
      {
            if((((Route*)routesFounded->get_data())->getNodeList())-
            >get_size()  <
            (retVal->getNodeList())->get_size() )
            retVal = (Route*)routesFounded->get_data();

      }
      return retVal;
}
return retVal;
}
```
Fig 15: “FindRoute” function

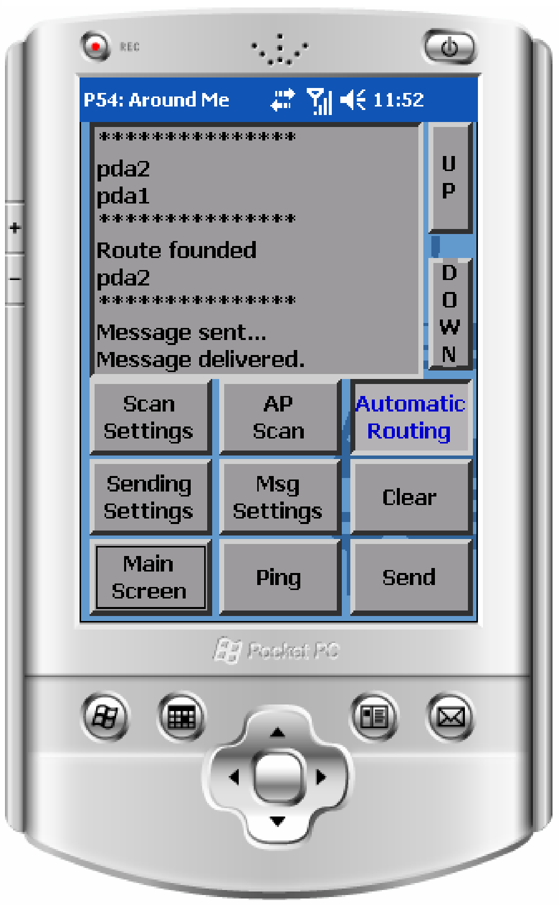
This function first tries to find a direct route (with no hops) to the destination. If there is no such route, then it will find the shortest one (with the smallest number of hops).

5.3.3 Sending Message

To send a message using automatic routing the user must press the “Automatic Routing” button on the main screen of the AroundMe application. In this way the “FindRoute” function will be called to find a suitable route for the message. The message body is then created for the P54V2 message that will be sent. The list of application names through which a message must pass on its way to the destination is extracted from the route object that was returned after the function call to “FindRoute”. This data is appended to the message body of the message. After sending the message, the application will wait a certain period of time for an answer. The next picture shows what will appear on the main screen of the application after clicking “Send”:



```
Possible routes:
pda1
pda2
****************
pda2
pda1
****************
Route founded
pda2
```



```
***************
pda2
pda1
***************
Route founded
pda2
***************
Message sent...
Message delivered.
```

Fig 16: Screen just after clicking “Send” **Fig 17:** Message is sent and delivered

5.3.4 Message Formats

The message body that is created during the sending procedure has the following format:

AUTOMATIC_SENDING%FROM_APP|FROM_APP|…%TO_APP|TO_APP|…%MESSAGE_TEXT

The first part of the message body is an indicator for the message handler of the receiving part. The second part is a list of character separated strings that represent application names of remote AroundMe applications through which the message passed on its way to its destination. At the source side this part has only one name – source application name – and as it travels to its destination it grows larger. Every application that receives this message adds its name to the end of this list to indicate that the message has passed through it. The third part of this message body is a list of character separated strings that represent the route for the message. When the message passes some of the nodes (applications) in this route, the node (application) will remove its name from the list. From this explanation you can see that in fact application names are migrating from the third part to the second part of the message body. The fourth part of this message body format is the message text itself. For example if PDA1 wants to send a message to PDA4 using PDA2 and PDA3 as routers, the evolution of this message will look like this:

1. AUTOMATIC_SENDING%PDA1%PDA3|PDA4%Some text – this message was received by PDA2
2. AUTOMATIC_SENDING%PDA1|PDA2%PDA4%Some text – this message was received by PDA3
3. AUTOMATIC_SENDING%PDA1|PDA2|PDA3%EMPTY%Some text – this message was received by PDA4

Note that in the first step there is no PDA2 name. The algorithm will take the first node from the Route object as the first destination application and appends only the application names to which the message must be sent from there.

In order to have the delivery report, the destination application creates a message that has the following format:

AUTOMATIC_REPLY%TO_APP|TO_APP|…

Every application on the route of this reply message removes its name from this list of names. At the beginning (on destination's side) this list is in fact the whole route that the message traveled on its way from the sender to the destination.

5.4 Access Points Scan

In order to perform an access point scan, the user must first choose the target PDA. First set the remote or host application name ("This PDA") into the "Scan" edit box in the "AP Scan Settings" window. Next, by clicking on the "AP

Scan" button the user will trigger the sending of the following message to the destination application:

AP_REQUEST

When this message reaches its destination, the message handler for this message will call the "getMACAndRSSIData" function. This function uses functions from the "GetRSSILib" library (3.2).

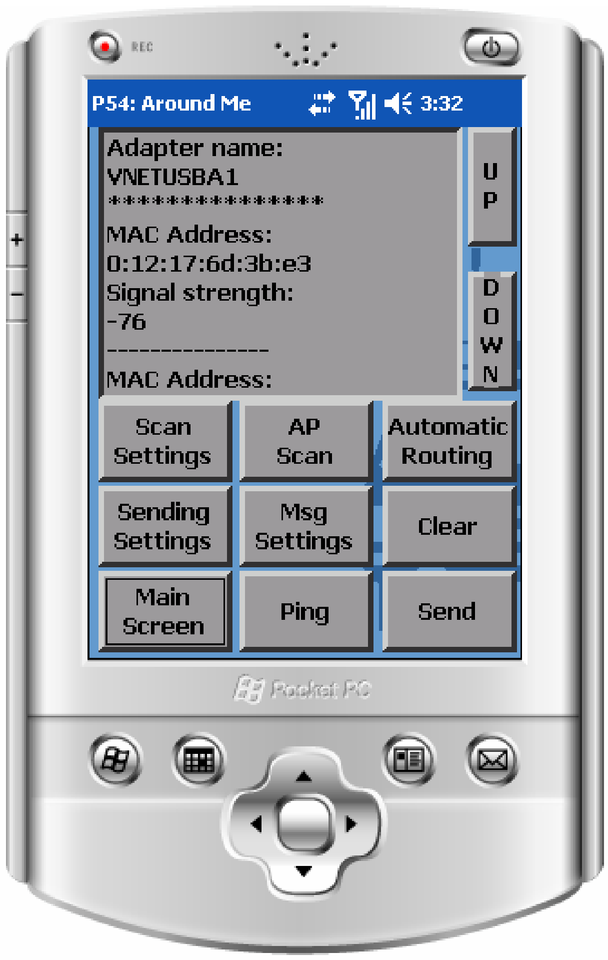
```
/**
      \fn getMACAndRSSIData()
      \brief Function scans for APs that this PDA sees and gets their
      MACs and signal strengths
      \return Returns instance of APDataType structure
*/
APDataType* getMACAndRSSIData()
{
      APDataType* ap = NULL;
      int iSuccess = 0;             // flag for scaning
      wchar_t wAdapter[32] = L"";   // name of adapter on local PDA
      wchar_t wMac[128];            // mac address
      wchar_t apData[1024]= L"";    // mac and rssi data

      /*
            allocate arrays for GetRSSILib functions - must do this
      */
      int macAddressArray[6*MAXAPS];
      int rssiArray[MAXAPS];

      for (int k = 0; k<6*MAXAPS; k++)
            macAddressArray[k]=0;

      for (k = 0; k<MAXAPS; k++)
            rssiArray[k]=0;
      int pnAPs;            //see how many MAC addresses the function found
      // gets name of WiFi adapter for local PDA
      GetEthernetAdapterNames(wAdapter, 32);
      wcscat(apData,wAdapter);// append adapter name to resulting data
      wcscat(apData,L"|");    // append separator
      iSuccess = Trigger(wAdapter); // triggers the AP scan
      if(iSuccess)
      {
            Sleep(850);       // Wait between 0.5-3 seconds before
                              scanning for signal strengths
            /*
                  get signal strengts and macs
            */
            iSuccess = GetRSSI(wAdapter, &pnAPs, MAXAPS,
            macAddressArray, rssiArray);
            if(iSuccess)
            {
                  /*
                        append all mac addresses and signal strengths
                  */
                  for (j=0; j<pnAPs; j++)
            {
                /*
                    first number of MAC address
                */
                // transform number into HEX value
                _itow(macAddressArray[(6*j)], wMac, 16);
                /*
                    compute other 5 numbers from MAC address
                */
                for(int i = 1; i<6; i++)
                {
                    _itow(macAddressArray[i+(6*j)],wTemp,16);

                    wcscat(wMac,L":");
                    wcscat(wMac,wTemp);
                }
                // append MAC address to resulting string
                wcscat(apData,wMac);
                wcscat(apData,L"|");
                _itow(rssiArray[j],wRSSI,10);
                // append RSSI strength to resulting string
                wcscat(apData,wRSSI);
                wcscat(apData,L"|");
            }
            ap = new APDataType;
            wcscpy(ap->macsAndRSSI,apData);
            wcscpy(ap->nameOfAdapter,wAdapter);
        }
        else
        {
            // failed to get RSSI strengths and MACs
            return ap;

        }
    }
    else
    {
        // signal strength failed
        return ap;
    }
    return ap;
}
```

Fig 18: Code of `getMACAndRSSIData()` function

This function returns APDataType which is a structure that keeps MAC (Media Access Control) addresses and RSSI (Received Signal Strength Indicator) data. Functions from “GetRSSILib” library must be called in a certain order:

1. GetEthernetAdapterNames – this function must be called first. It returns device names of Ethernet adapters. Hopefully it will get only one and hopefully it will be an 802.11 adapter.
2. Trigger – this function triggers access points signal strength scan for adapter name returned by the previous function.
3. GetRSSI – read RSSIs from the network card

Data is collected in this function and is stored in an instance of the APDataType structure, and then it is returned to the application which asked for this data through a P54V2 message and its message body. This message has a message body that begins with the indicator AP_RESPONSE and is followed by data from the structure. When the message reaches the message handler on the receiver side, the data is displayed on the screen. The data received from the remote application is stored in a list as well as data collected by scanning from the PDA that initiated the access point scan on the remote PDA device. Data from these two lists can be used in future developments to calculate an approximation of the distance between the two PDA devices.



Adapter name:
VNETUSBA1
****
MAC Address:
0:12:17:6d:3b:e3
Signal strength:
-76

MAC Address:

Fig 19: MAC addresses and signal strengths displayed after access point scan

6. Installation Manual

There are several steps to successfully installing the AroundMe software on a PDA computer. This is an example of an installation for the AroundMe application whose name is “pdaaroundme”:

1. Install the Project54 platform for PDA devices on the PDA by double clicking on the Project54PocketPC setup file. After the installation wizard is opened, follow the instructions of the wizard. AroundMe is not part of the default Project54 installation so you will have more to do to install it on your PDA.
2. Manually add AroundMe.dll and GetRSSILib.dll to the Project54PDA folder of the PDA device.
3. Edit the allapps.txt configuration file. In this file you have to add next line:
```
AroundMe   {3feb151f-0c54-11db-9606-00e08161165f}  pdaaroundme  2,1
AroundMe Finds and communicate with pda's around
```
   Note that every application that is part of an ad-hoc network must be named differently. In this example it is “pdaaroundme”. So, on every PDA device this line will differ by the name of the application.
4. Edit the P54Components.txt configuration file. In this file you have to add the following lines:
```
//
////////////////////////////////////////////////////////////////////
//
// AroundMe DLL
{HKEY_CLASSES_ROOT\CLSID\{3feb151f-0c54-11db-9606-00e08161165f}}
            []                  Car54, AroundMe
      <InprocServer32>
            []                  \Project54PDA\AroundMe.dll
            [ThreadingModel]    Free
      <ProgID>
            []                  Car54.AroundMe.1
      <VersionIndependentProgID>
            []                  Car54.AroundMe

{HKEY_CLASSES_ROOT\Car54.AroundMe}
            []                  Car54, AroundMe
      <CLSID>
            []                  {3feb151f-0c54-11db-9606-
00e08161165f}
      <CurVer>
            []                  Car54.AroundMe.1

{HKEY_CLASSES_ROOT\Car54.AroundMe.1}
            []                  Car54, AroundMe
      <CLSID>
            []                  {3feb151f-0c54-11db-9606-
00e08161165f}
```
   These lines are the same for every installation.
5. Edit the registrysettings.txt configuration file. In this file you have to add several lines:
   - In the “Application Manager” module under “Applist” optional parameter group name add:
```
[pdaaroundme]     {3feb151f-0c54-11db-9606-00e08161165f}
```

- In the “Main Screen” module under “Applist” optional parameter group name add:
  ```
  [pdaaroundme]                 2,1 Around|Me
  ```
  This represents the position in the button grid of the “AroundMe” button on the main screen and caption of the button.
- Add a new module for the AroundMe application:
  ```
     // the AroundMe application
  {AroundMe}
     <Messaging>
           [self]             pdaaroundme
           [Appmanager]       pdamainscreen
           [aroundme]         pdaaroundme
           [aroundmeV1]       pdaaroundme1
           [aroundmeV2]       pdaaroundme2
           .     // other remote applications
           .
           .
           [RecordsQueries]  rqueries
           [Scanner]          pdascanner
           [Logger]           pdalogger
           [SpeechIO]         pdaspeechio
           [ButtonControl]   pdabuttoncontrol
  ```
  Parameters named aroundmeV1, aroundmeV2, ….aroundmeVn are in fact aliases for remote applications that are known to this AroundMe application.
- In the “Proxy” module add lines for every remote application that is known to this AroundMe application. Lines are consisted of the name of the remote application, its IP address and access port.
  ```
  [pdaaroundme1]   192.168.0.1,3054,c
  [pdaaroundme2]   192.168.0.2,3054,c
  …
  ```

6. In the initialization folder run the “Proxy Configuration” application. In this application you have to add the IP addresses of all the remote applications that are known to this AroundMe application and with which this application can communicate.
7. Run the InitComponents and RegInit applications to update registry with the new settings.
8. Now you can run Project54 and the AroundMe software.

7. Testing

Testing of this project had several phases. During development software was tested and debugged at all times. Testing in the laboratory was most common and it helped discovering many bugs and errors in the software. As this system is designed to be used outdoors, most tests were outdoor tests. Complete test procedures for testing is part of the documentation for this project and can be found in the /doc/TestProcedure folder. The document describes in detail setup, tests and expected results. In most of the cases the software met the expectations of tests. On the other hand there were some unexpected behaviors of the software and system as a whole. During one outdoor test the PDA that was in the middle (situation is described in TestProcedure.doc) couldn’t “see” the other PDAs even though it was far from being out of range of the other PDAs. In some cases the ad-hoc network seems to block and as a result the ping action will be unsuccessful and the user can conclude that some of the PDAs are out of range. Strength of radio signals emitted from Wi-Fi antennas changes over time. This can also produce some strange behavior. For example, during one of the tests, two PDAs were far enough apart that they could not see each other but they could see the middle PDA. Routing of messages was needed in order to send a message between the two distant PDAs. The sending action was successful (person holding destination PDA raised a hand as a signal that he received a message), but a delivery report didn't reach the source PDA. When this test was repeated it was a complete success. This situation led to the conclusion that the signal strength in the ad-hoc network had something to do with the results of the first attempt.

Testing in the building also produced some strange results. The aim was to distribute PDA devices around the building and to try to go behind various obstacles and communicate with the other PDA devices. Similar things involving variation of signal strength happened during tests in the building. More testing of this system is needed in order to discover some possible errors. One thing that can be done in the future is to log all events that have something to do with using the wireless network. Data collected in this way can be statistically interpreted and some useful conclusions can be drawn for future developments.

8. Problems

The problems encountered during the development and testing phases of this project are:

- After an access point scan is performed on some PDA, the network card acts like it is blocked. The user can not send a message from that PDA, but can receive messages. Also, pinging a PDA that performed an access point scan will be unsuccessful because that PDA will not be able to send a reply message that it is reachable. The reason for this bug is unknown. After restarting the whole Project54 application the problem disappears.
- After starting the system and performing a ping for the first time ping data will not be accurate. For example if we have a system consisting of three PDA devices pinging from PDA1 we will get following data:

  ```
  PDA1->PDA2
  PDA1->PDA2->PDA3

  PDA1->PDA3
  PDA1->PDA3->PDA2
  ```

  The first and second lines are not necessary. After repeating the ping action the user will get the expected output:

  ```
  PDA1->PDA2->PDA3

  PDA1->PDA3->PDA2
  ```

- Sometimes a wireless card stops working and the user will have to reset the whole PDA device in order to make it work again. This I noticed on the Toshiba PDA devices but not on the HP PDAs. So, my conclusion is that this has something to do with the wireless adapters installed on the Toshiba PDA devices.

9. Suggestions for Future Developments

From this point of the project many further developments are possible. Some of them are:

- At this point the ad-hoc network is maintained and created manually. It will be very useful if this network can be created automatically after two or more PDA devices are in short range. For example from the introduction, if two or more police officers are in close proximity a network could be established. Other police officers coming to the crime or accident scene can join this new network. They will be offered to join the network and make them visible to others and also available for communication.
- At this point only a version for the PDA exists. In the future, this software can be ported to embedded computers in the cars. This version can be improved by adding server functionality. For example, a computer in the car, that arrives at the accident or crime scene first starts acting as a server (or access point) making it possible for PDA devices to connect to it and communicate through this computer.
- The computer from the example above can be used for measuring proximity between PDA devices in surroundings (something similar to NearMe project [1]) using signal strength measurements.
- Some aspects of the Proxy application can be improved. For example, in this version the proxy is using registrysettings.txt as a configuration file. From there the application can read the names and IP addresses of other applications with which it can communicate. Every new application must be added manually to this file. More automation for this is needed because police officers will not have time, nerves or knowledge to manually setup everything that is needed for successful communication between two or more PDA devices. Using an XML format for configuration files is more flexible for on-the-fly changes.
- There are only a few widgets offered for building the GUI of the application. More widgets are needed for building intuitive and more complicated GUIs.

10. Development tools

Tools used for software development in this project are:

- MS Embedded Visual Studio 4.0

- MS Embedded Visual Studio 4.0 Service Pack 3
- MS Active Sync 4.1
- MS Pocket PC 2003 SDK
- MS Visual Studio 2005
- Doxygen – for code documentation

Some drawbacks were noticed during the development of software using these tools:

- The compiler in Embedded Visual Studio 4.0 was acting strange in some cases. It reported errors on certain lines of code even though the code was good.
- If you try to compile code from Project54 using the emulator in the configuration (in both options – debug and release) it will report "unresolved external symbol" for functions exported from GetRSSILib.lib
- MS Pocket PC 2003 SDK makes windows disable some drivers (because windows consider them to be threat for system stability). You must fix this problem in order to run the emulator from this SDK. On the other hand that emulator can not run the Project54 software.
- Even though the emulator from MS Visual Studio 2005 is much better than the one from the MS Pocket PC 2003 SDK you cannot emulate a wireless card on it. There is a possibility to connect the emulator to a wireless card installed on the desktop computer that is running VS 2005, but it doesn't work as it supposed to work. On the other hand, installing and handling the Project54 and AroundMe software on this software is exactly the same as on a real PDA device.

11. Conclusion

The system that I built during last few months is not really big but it is one whole working system that can be improved in many ways. It opens many questions and leaves a lot of room for future projects and research. All the steps that I planned were completed successfully.

Finding documentation about similar projects was not a difficult task. There are a handful of projects similar to AroundMe. After learning about Project54 and examining the code of applications included into this framework I found all the elements that I needed to build a simple test application. By building a test application I have learned how to use the Project54 framework and some of the applications included in it. With this knowledge I was ready to start with the main project. This final goal was accomplished.

More testing is needed in order to find out all the characteristics of this system. I tried to write code as understandable as possible so other people, who will use this as a base or example, can easily understand it. I also used doxygen to generate documentation for functions, classes and structures. This documentation is in HTML form, which makes it easy for navigating.

As I underlined earlier in this text, this is a basic application for message exchanging between PDA devices – a prototype. The protocol can be improved and made more efficient and sophisticated. New functionalities can be added. There is potential for turning this prototype system into a part of the Project54 framework.

12. References

1. John Krumm and Ken Hinckley, "The NearMe Wireless Proximity Server"
2. Gerd Kortuem, Christian Kray and Hans Gellersen, "Sensing and Visualizing Spatial Relations of Mobile Devices"
3. Alessandro Genco, Salvatore Sorce, Guiseppe Reina and Giuseppe Santoro, "An Agent-Based Service Network for Personal Mobile Devices"
4. David Garlan, Daniel P. Siewiorek, Asim Smailagic and Peter Steenkiste, "Toward Distraction-Free Pervasive Computing"
5. W. Thomas Miller, III, "Remote Project54 application messaging via the Proxy Application"

12.1 Other Literature

1. Seng W. Loke, "Context Aware Artifacts – Two Development Approaches"
2. W. Keith Edwards, "Discovery Systems in Ubiquitous Computing"
3. Daniel Wagner, Thomas Pintaric, Florian Ledermann, Dieter Schmalstieg, "Towards Massively Multi-User Augmented Reality on Handheld Devices
4. Jay Summet, Rahul Sukthankar, "Tracking Locations of Moving Hand-Held Displays Using Project Light"
5. Florent Rivreau, "A Prototype Home Automation Project Using the Project54"
6. W. Thomas Miller, III, "Project54 client/server application messaging"
7. Fluke Networks, Wireless Site Survey Best Practices