PocketDrive: A System for Mobile Control of Desktop PC and its Applications Using PDAs

Yenel Yildirim and Ibrahim Korpeoglu
Department of Computer Engineering
Bilkent University
06800 Ankara, Turkey

Abstract— **Today, consumer electronic devices and PCs are inevitable parts of our daily life. Controlling those devices remotely is an important aspect of the technology. We have already universal remote control devices for controlling consumer electronic devices. Similarly, we may control our desktop and laptop PCs and their applications remotely via portable and smaller computers like PDAs and Pocket PCs. This paper presents a system and its architecture that enable a wireless-enabled PDA to control a PC and its applications remotely over a 802.11 or Bluetooth link. With such a system running on a PDA, a user can start, run and control PC applications from any location that is reachable via 802.11 link. This enables flexibility, ease of use, and freedom for the user of PC applications.**

I. Introduction

Consumer electronics devices and personal computers became inevitable part of our life. Similarly, mobile devices and computers like Pocket PCs are becoming more and more commonly used in our daily life. Controlling consumer electronics devices and computers remotely is an important as aspect of the technology. Today, we have universal remote control devices to control consumer electronic devices such as TV sets. Similarly, it is desirable to remotely control stationary desktop/laptop PCs and their applications. For example, an instructor in class-room may want to control a PowerPoint presentation running on his laptop computer and projected on the screen using a PDA (i.e. pocket PC) or cell phone, while he is moving around in the class-room freely. In this way, he does not have to go to the laptop each time when he wants to update the PowerPoint screen.

This paper is about how to control PC applications remotely inside a house, an office, or a conference room. As the remote control device, we consider using a general purpose pocket PC computer with wireless LAN or PAN capability (i. e. supporting WiFi or Bluetooth). For this purpose, we designed and implemented a mobile system software, called *PocketDrive*, that enables a Pocket PC to act as a remote controller device for desktop/laptop PCs and their applications. In this paper, we introduce *PocketDrive*, its architecture, and how we have realized it.

PocketDrive is a system/tool that allows users to control their desktop computer applications from a PDA over a wireless network or Internet. Its architecture is based on client-server paradigm. It consists of two parts: a server part and a client part. The server part runs on a desktop PC (or laptop PC depending on the usage scenario) to be controlled remotely. The client part runs on a mobile Pocket PC device that can be easily carried by a user and that will act as the remote controller device for desktop PC and its applications.

The server side of the system is capable of listening incoming connections, sending and receiving data, processing control commands, taking screenshots, modifying and sending images back to client side, and sending mouse and keyboard events to the operating system.

The client is capable of opening connections to the server, sending/receiving data and processing control commands, changing the view area (which we call as viewport); and sending mouse and keyboard events to the server side.

Both the server side and client side of the system are coded in C#.NET, and .NET Framework 1.1 is used as the development kit. The server side of the system runs on top of the Windows XP operating system [1] and the client side runs on top of the Windows Mobile 2002 operating system. [2]

We identified the following as the requirements of a system that enables a Pocket PC to be used as a remote and mobile control device for desktop PC applications. Those identified requirements helped us as the basic guidelines in designing our system.

- *Ease of use*: The system should be easily launched, configured and used. It should have a nice and graphical user interface.
- *Mobility*: The system should support mobility of the user while controlling the desktop computer application. Mobility can be enabled if the remote control device is portable and if its connection to the desktop computer is wireless. The wireless connection can be a short-range local or personal area connection, most of the time. In that case, the roaming range can be up to 100 or 300 meters depending on the wireless technology used and on the propagation environment.
- *Flexible Control*: A user should be able to control and execute as much functionality as possible. It is the best if the user can do everything that he/she can do on a desktop computer also in a remote manner. The user should be able to give keyboard inputs and mouse inputs to the desktop PC and also should be able to get as much screen information as possible. (If applicable, displaying the whole screen area is preferred.)

This work is supported by The Scientific and Technical Research Council of Turkey (TUBITAK) with project EEEAG-104E028.

- *Power*: The system should be power efficient since the PDAs are power constrained devices and have limited energy.
- *Reliability*: The system and connectivity should be reliable enough so that a user can control the desktop computer without losing data and/or commands.
- *Bandwidth*: The system should be bandwidth efficient on the wireless link between the PDA and desktop PC, since the same link can be used for many other applications that are run at the same time, like an FTP transfer, a web browser activity, a backup activity, and so on.
- *Enabling Feedback*: While interacting with the desktop PC using a mobile PDA, the user should get enough feedback from the system about what is going on and about the status of the executed operations.
- *Asymmetric Functionality*: A desktop computer has advantages in comparison to a Pocket PC in terms of computation power and unconstrained energy sources. Therefore the system design should be asymmetric whenever possible, giving more computational overhead to the server side of the system than the client side. In other words, a thin client model is a preferred model for the architecture of the system [12].
- *Security*: Not everybody who has a Pocket PC in the vicinity should be able to control a desktop PC and its applications. Only authorized users should be able to do that. Therefore the system should support authentication and authorization of the users that would like to access and control the system remotely via wireless PDAs.

Throughout the paper we will use the terms “pocket PC” and “PDA” interchangeably to refer to the same type of hand held device. The rest of the paper is organized as follows. In the next section, Section II, we briefly describe some related work. In Section III we describe the PocketDrive system’s architecture and design, and its implementation. And in Section IV, we give our conclusions and discuss some future work items.

II. Related Work

There are already some products for controlling a PC from a PDA, similar to our work. Some of them are Remote Control PocketPC [10] and Mobile Administrator [11].

The Remote Control PocketPC system described in [10] uses also client/server paradigm and provides access to a PC over a TCP/IP connection that goes over either a modem or a wireless channel. It provides mouse and keyboard control after secure access granted with correct username/password combination. These features are similar to our PocketDrive system; but the system in [10] lacks the display capability we offer as at it supports only grey scale, 16 colors and 256 colors resolution whereas our system can display 16-bit PNG screenshots of the PC screen. Additionally, PocketDrive supports zooming and presentation mode with user-friendly GUI for fast forward and backward jumping on a presentation.

The Mobile Administrator [11] system enables the remote control of a computer using a portable device. This tool supports running commands on the server side, but it does not allow mouse and keyboard control, and it does not show screenshots of the PC on the Pocket PC screen. With our PocketDrive system, a user is able to control keyboard of a PC, and in this way the user can open a command window and run any command on the PC. Additionally, PocketDrive can show the current state of the PC screen on the PDA screen.

III. System Architecture and Design

In this section, we will explain the architecture and design of the PocketDrive system, including the internal components. We will also discuss some optimization issues.

A. Network Connectivity

The connection between a controlling PDA and a desktop PC can be over a single wireless hop, or over many wired and wireless hops using the Internet infrastructure. In both cases IP (Internet Protocol) is used as the communication glue and the routing protocol, and TCP is used as the transport protocol.

Figure 1 shows the connection of a PDA to a desktop computer using a single hop wireless link. The wireless link technology can be a wireless LAN technology such as IEEE 802.11 a/b/g [3], [4], or a wireless PAN technology such as Bluetooth [6], [7]. For such a direct connection, both PDA and desktop PC should be wireless capable. The PC can have a wireless adaptor to talk to the PDA or it can first talk to a wireless access point over Ethernet and then the wireless access point can bridge this connection to the PDA over the wireless link.

Whether the wireless link between the PDA and PC is 802.11 or Bluetooth, the rest of the protocol stack can be the same. We can run TCP/IP on both Bluetooth link layer and 802.11 link layer. The PocketDrive system programs run over TCP/IP, hence are not affected by the link layer technology. Of course the performance and some other metrics are affected. 802.11 provides better throughput and range, whereas Bluetooth provides better energy efficiency [5], [8]. Additionally, if Bluetooth is used as the link layer, TCP/IP is not required. Using the RFCOM sublayer of Bluetooth, the PocketDrive software can be implemented so that it uses serial communication using RFCOM for message exchange between PDA and PC. In this case, however, a reliability layer has to be included as part of PocketDrive software, since the RFCOM sublayer of Bluetooth and the layers below does not guarantee hundred percent reliability for the delivery of the bytes and packets [7], [8].

When we use IP as the routing layer and TCP on top of it, the connection between the PDA and PC does not have to go over a one-hop link but can be any number of hops. The PDA and the PC can be located anywhere on the Internet. Hence the connection can be over the Internet. This scenario is shown in Figure 2. In this case, the PDA can be connected to an access point via a wireless link and the access point is connected to the Internet. Similarly, the desktop PC is also connected to the Internet via a wired and wireless link. The

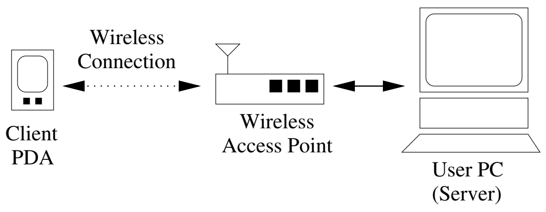


Fig. 1. Network connectivity underlying the PocketDrive system. It uses short-range (local or personal area) wireless technology for the connectivity between the PDA and the desktop PC.

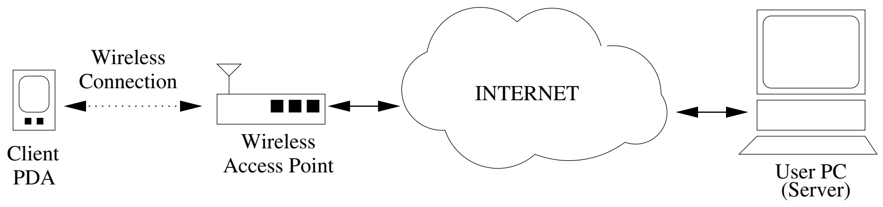


Fig. 2. Connectivity between the PDA and desktop PC over Internet. In this case, only the last hop to the PDA is wireless, and the PDA user can control the desktop PC applications from any point in the Internet no matter how far it is to the desktop PC.

PocketDrive system can then run on top of this configuration without requiring any modification.

B. Operation of PocketDrive

PocketDrive system consists of two software programs: one server program running on a desktop PC or laptop PC that is to be controlled remotely; and one client program running on a PDA to be used as the remote controller.

The server process starts listening on a well-known TCP/IP port after being started up on the desktop PC. The PDA user has to know the IP address of the desktop PC and the well-known port number for being able to connect to the desktop from the PDA. Additionally, a username and password is required for the user to login to the system. The server process asks the client process a username and password, and then authenticates the user. In this way, only users with previously created accounts can access the PC for controlling. These parameters (server IP address, port number, username, password) are maintained at the server side and are configurable via a GUI interface.

A PDA user that would like establish a connection to the PC for controlling the PC applications enters the required information (IP address, port number, username and password) and a TCP connection is established to the server. Over the TCP connection, besides sending username and password, the client process also sends some parameter values regarding the screen width and height. If the server authenticates the user, it replies back with a *login* command and *OK* message. Otherwise, if the authentication is not successful, the server replies back with a *login* command and *NO* message. The TCP connection is closed in this case.

If the client gets logged in to the server successfully, other operations can be executed and controlling of the PC can start. For this, the client first requests the current state of the PC screen. It sends a *Send New Image* command to the server to obtain the screenshot of the desktop PC. Upon receiving such a command from the client, the server captures the image of the screen, resizes it according to the dimensions of the client's screen, and makes the color depth 16 bits. Then the server sends *Image* command to the client including the size of the image in bytes, and the binary data corresponding to the image. When the client gets the *Image* command, it reads the binary image data and creates an image to be displayed on PDA's screen. The image corresponding to the screen state of the PC is then displayed on the screen of the PDA as part of the GUI structure for the PocketDrive client.

After getting a screenshot from the PC, the PDA can request more screen shots. For this, it should again send a *Send New Image* command to the PC. This mechanism, i.e. waiting for an image to be downloaded before sending the next request for another image, is a simple flow control mechanism between the PDA and PC. Another alternative could be requesting images periodically; but then the period should be adjusted depending on the speed of PDA and the bandwidth of the wireless link. The mechanism that we use is adaptive to the changes in the wireless link properties and available bandwidth.

After getting the screenshot of the PC, the PDA user can start doing some control operations. Those include mouse operations and keyboard operations.

A mouse operation is triggered when the PDA user touches to the screen of the PDA that has the PC image. Additionally, the PDA user has a GUI through which he/she can indicate whether she would like to emulate a left mouse button press event or a right mouse button press event. Then the mouse operation and location is transported to the server which converts the mouse position to a point on the PC screen. Then the server sends a mouse event to the PC operating system and appropriate action is executed on the PC. Similarly, a keyboard operation can be triggered when the user presses on a key on the PDA keyboard. The appropriate keyboard command is conveyed to the server which in turn sends keyboard events and character codes to the PC operation system. Besides these, if an error occurs on the server side, an *Error* command and an appropriate error string is sent to the client side.

If the client gets disconnected from the server unexpectedly, the client tries to reconnect to the server if the user has enabled the “try to protect disconnections” option. Otherwise, reconnection is not attempted and the user is informed about the disconnection.

C. Control Commands and Data Flow

Table I shows the control commands that are used between the PocketDrive server and client. A command is sent in a message and represented with a string followed by a new line character. The commands *Login*, *Error*, *Okey* and *Not Okey* are used at the beginning of client server handshake and authentication. The *Error* command indicates an error condition and is used in other phases as well.

The *Left Click*, *Left Double Click*, *Right Click*, *Right Double Click*, *Mouse Left Down*, *Mouse Left Up*, *Mouse Right Down*, *Mouse Right Up*, *Mouse Move* commands are mouse related commands and are used to control the mouse pointer on the desktop PC screen. The relative location of the pointer on the PDA screen is sent to the server side and the server calculates the actual location on the desktop screen.

Key Down and *Key Up* commands are keyboard related commands and are used when the user would like to provide keyboard input to the PC applications via the PDA keyboard panel. For that, the character code of the key is sent inside a command to indicate which key is pressed.

Since the screen size of a PDA is much smaller than the screen size of a desktop PC, the screen view of the PC has to be scaled down, and this may cause a low quality image to be presented on the PDA screen. Therefore, supporting *zooming* of the screen image is an important feature to have on the client side. The PocketDrive supports this in a simple manner. The image shown on the PDA screen is divided into four quadrants, and each quadrant can be zoomed to occupy the whole PDA screen. For that, the client side issues *Change Screen Zoom* command including the information about which part of the screen has to be zoomed: left top, right top, left down, or right down.

Command Name	Command String
Login	$lg
Error	$err
Okey	OK
Not Okey	NO
Send New Image	$sni
Image	$im
Left Click	$lck
Left Double Click	$ldk
Right Click	$rck
Right Double Click	$rdk
Mouse Left Down	$mldn
Mouse Left Up	$mlup
Mouse Right Down	$mrdn
Mouse Right Up	$mrup
Mouse Move	$mmv
Key Down	$kdw
Key Up	$kdu
Change Screen Zoom	$cha

TABLE I
CONTROL COMMANDS AND THEIR STRING REPRESENTATIONS.

D. Structure of the PocketDrive Client

Figure III-C shows the internal software structure of the client side and server side of the PocketDrive system. Each side is divided into components with specific well-defined functionality. The components constituting the client side are: Client GUI, Saver/Loader, Command Handler, and Connector.

The *client graphical user interface (GUI) component* is responsible for interacting with the PDA user. With this interaction it can get the server IP address, port number, username, password, and the option if disconnection will be protected or not. Those are the information required to login into and authenticate with the server. Another function of the GUI component is to show the screenshot images arriving from the server side. This function is vital for enabling the PDA user to have total control on the PC and to keep track of what is going on. Another responsibility of the GUI component is to detect mouse and keyboard events, and send those events to the server side via the help of the command handler component. The GUI also has a presentation mode which is suitable to easily direct PowerPoint presentations. In this mode, the user can send *Forward* and *Backward* commands to the PowerPoint software running on the desktop or laptop computer without getting updates about the desktop screen. This saves wireless bandwidth and helps the remote commands to execute faster.

The *saver/loader component* is responsible for saving the settings (server IP address, port number, username, password and “try to protect disconnection” option) to a file, called `settings.ini`, and loading these settings when the program re-starts.

The *command handler component* is responsible to send commands through the connector component and analyze/process incoming commands. For example, when it receives an incoming *Image* command, it first reads the size of the image and then reads that many bytes from the connector component in binary form. After then, it creates the image and gives the image to the GUI component to be displayed on the PDA screen. The command handler component also detects disconnections via the *null* command and informs the GUI about disconnections.

The *connector component* is the layer that provides communication with the server side. It sits on top of the TCP/IP transport layer. It is responsible for initiating the connection and interacting with the server. It uses TCP as the transport layer, therefore the communication is stream oriented, having no packet boundaries at the transport layer. The packet boundaries are detected at this component. The packets include commands and those commands are given to the command handler component. Similarly, commands are received from the command handler component and sent through the TCP layer to the server side. This component is also responsible for tearing down the connections when required.

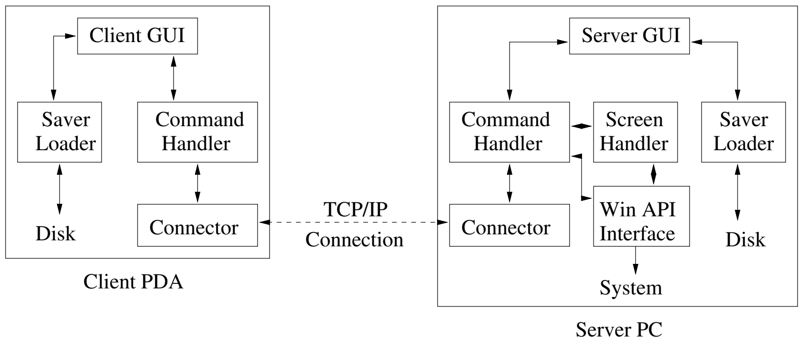


Fig. 3. Internal Structure of PocketDrive Server and Client Software

E. Structure of the PocketDrive Server

Figure III-C also shows the structure of the server side software that is to be run on a desktop or laptop computer. The server side is composed of the following components: Server GUI, Saver/Loader, Command Handler, Screen Handler and Win API Interface, and Connector.

The *server graphical user interface (GUI) component* is responsible for interacting with the user at the server side and providing the user the ability to set the values of some parameters such as listen port, username and password. Another functionality of the GUI is to present logs which provide useful information about connections.

This *saver/loader component* at the server side is very similar to the corresponding component at the client side. It is responsible for saving the settings (port, username, password and logging option) to a system file (`settings.ini`) and loading the saved data when the program starts. In order to protect the saved information from other people who should not access the system, the 128-bit DES encryption method is used to encrypt the information [9].

The *command handler component* sends the commands through the connector components towards the client side, and receives and processes the commands arriving from the client side. As part of processing of the incoming commands, it uses Win API interface in order to send mouse and keyboard events to the desktop operating system. Additionally, it uses the screen handler component for capturing the screen state. This is done when the client side issues a *New Image* command to the server. Consequently, the server side first gets a screenshot with the help of the screen handler component, and resizes the image according to the client's screen size and viewport (view area). The color depth is decreased from 32 bit to 16 bit and the image is sent towards the client as part of the *Image* command in PNG format.

The *screen handler component* is responsible for capturing the current state of the screen and providing the captured screenshot images to other components.

The *Win API Interface component* is responsible for sending mouse and keyboard events to the desktop operating system. It also provides screen capturing functions to the screen handler component. It uses some dynamic link libraries (dlls) for this purpose [13].

The *connector component* is responsible for listening to incoming connections on a well-specified port, and accepting the incoming connection requests arriving from the client side. Similarly to the client side connector component, it runs over TCP/IP, and uses stream-oriented communication while talking to the corresponding component in the client side. It is responsible for converting the stream oriented bytes into packets (commands), and vice versa. The command packets are passed to the command handler component, or the command packets are received from the command handler. This component is also responsible for termination of connections.

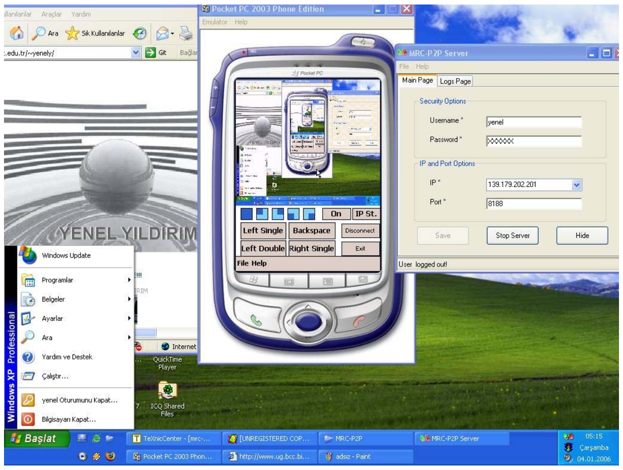
F. Performance Issues and Design Choices

During the design phase, we chose TCP as the transport layer instead of UDP in order to have reliable delivery of commands and data. Additionally, we use IP addresses to address the end points [14], [15]. This enables the system to work also over Internet, i.e. the client and server can be located in any two points on the Internet. If the system is to be used only locally, then use of IP addresses is not mandatory. Even use of TCP/IP is not mandatory. We can just use serial communication between the client and server.

In order to decrease the bandwidth usage during the transfer of commands and data (images), the client sends its screen height, width and interested viewport (all, left top, right top, left bottom, right bottom) information to the server, and the server stores this information and uses it while sending screenshots back to the client. When the server is ready to send a screenshot to the client, it first resizes the image according to client's screen parameters and the desired viewport, and also reduces the color depth from 32 bit to 16 bit. Then the image is sent to the server in PNG format.

To increase the performance further for applications like PowerPoint, the client has a presentation mode which only enables the user to control the keyboard and mouse without getting screenshot images from the server. This saves bandwidth and reduces the command execution latency.

A PDA has significantly limited computing power than a desktop. Therefore, a desktop computer should not overwhelm the PDA with commands and data. This issue comes up when the screenshot images have to be transferred from the server to the client. If the server constantly takes screenshots and transfers them to the client, the client may not be



Username * yenel

Password * XXXXXX

IP * 139.179.202.201

Port * 8188

User logged out!

Fig. 4. Screenshot of PocketDrive system

powerful enough to handle this. Tests show that, for example, 30 fps is too high to process for the PDA, and consumes too much wireless channel bandwidth. Therefore we need a feedback mechanism about the availability of the client for further images. This is achieved by sending the images from the server to the client upon demand. For this, the client sends *New Image* command to the server to request a new screenshot. The client does not issue another *New Image* command unless it receives a response, i.e. the screenshot, to the previous request. When it has received the screenshot, it can send another request for a new screenshot. This is like the ACK mechanism, or the Stop-and-Wait mechanism used in the transport layer protocols or MAC layer protocols. Via this adaptive mechanism, the refresh rate (frame per second) of PDAs will depend on their computational power and on the available wireless channel bandwidth.

IV. Conclusion and Future Work

This paper introduces our system for local or wide area remote controlling of desktop/laptop PC applications using handheld devices like Pocket PCs.

The contribution of this paper is providing the architecture and detailed design of a general purpose remote control system for controlling desktop PC applications from handheld devices. We also identified some general requirements of such a system and tried to meet those requirements as much as possible in the design and implementation of our system. The system is fully implemented and can be downloaded from one of the following URL addresses:

- http://www.mycoolsoftware.com/pocket.html
- http://www.cs.bilkent.edu.tr/∼korpe/lab/software/pocketdrive.rar

New features can be added to improve the security and usability including:

- Server scanner that scans and shows PocketDrive servers on a local area wired or wireless network into which a handheld device can get connected.
- Commands and data can be sent after getting encrypted for better privacy and security.
- New user types can be created with the different security levels such as a guest user who only observers the activities on the server.
- The size of a screenshot image can be reduced further by an appropriate compression scheme or image format. This will decrease bandwidth usage on the wireless channel.
- The same approach can be applied for mobile phones to use them as remote control devices as well.

Acknowledgements

We thank TUBITAK (The Scientific and Technical Research Council of Turkey) for supporting this work with project EEEAG-104E028.

References

[1] Winwdos XP operating system. "http://www.microsoft.com/windowsxp/default.mspx"
[2] Windows Mobile operating system 2002. system. "http://www.microsoft.com/windowsmobile/default.mspx".
[3] IEEE 802.11 Working Group, "http://grouper.ieee.org/groups/802/11/"
[4] WiFi Alliance, "http://www.wi-fi.org/"
[5] Matthew S Gast, 802.11 Wireless Networks: The Definitive Guide, 2nd Editon, O'Reilly, 2005.
[6] Bluetooth special interest group "https://www.bluetooth.org/"
[7] Bluetooth/IEEE802.15.1 specification document, "http://standards.ieee.org/catalog/olis/lanman.html#wirelessPAN"
[8] Brent A. Miller, Chatschik Bisdikian, "Bluetooth Revealed: The Insider's Guide to an Open Specification for Global Wireless Communications", 2nd Edition, Prentice Hall, 2001.
[9] Charlie Kaufman, Radia Perlman, Mike Speciner, "Network Security: Private Communication in a Public World", Second Edition, Prentice Hall, April 15, 2002.
[10] Remote Control PocketPC "http://www.bitween.com/sito/catalogphp?model=29&but=1"
[11] Mobile Administrator. "http://www.mobileadministrator.com/html/mobile_administrator.html"
[12] Thin-Client Server Computing. "http://members.tripod.com/ peacecraft/infomining/thinclnt.pdf"
[13] Windows MSDN Library. "http://msdn.microsoft.com/library/"
[14] Larry L. Peterson, Bruce S. Davie, "Computer Networks: A Systems Approach", 3rd Edition, Morgan Kaufmann, May 2003.
[15] W. Richard Stevens, "TCP/IP Illustrated, Volume 1: The Protocols", Addison-Wesley, 1993.