MOBILE PHONE CONTROLLED ROBOT-I

ANDY BOON TEEN YEN

A project report submitted in partial fulfilment of the
requirements for the award of the degree of
Bachelor of Engineering (Hons) Electrical and Electronic Engineering

Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

April 2011

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : _____

ID No. : _____

Date : _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"MOBILE PHONE CONTROLLED ROBOT-I"** was prepared by **ANDY BOON TEEN YEN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons) Electrical and Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor : Dr. Lo Fook Loong

Date : _____

ACKNOWLEDGEMENTS

I feel profound pleasure in bringing out this report for which I have to learn from different view of programming and use it to interface with hardware. I would like to thank everyone whom we had long discussion and without which it would not have been possible. First of all, I must express my gratitude to my respected research supervisor, Dr. Lo Fook Loong for his invaluable advice, guidance and his enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my loving parent and friends who had helped and given me encouragement.

Last but not the least I expresses my sincere thanks to Universiti Tunku Abdul Rahman for providing such opportunity for implementing the ideas of my own.

MOBILE PHONE CONTROLLED ROBOT

ABSTRACT

Many scenarios involve a robot to be controlled by a mobile phone. There are many ways to control a robot using mobile phone, for example IrDA (Infrared Data Association, WiFi, Mobile packed services (e.g. GPRS/EDGE/UMTS/HSPDA) or Bluetooth. The interface between the mobile phone and microcontroller is the subject of considerable discussion. Decision is made to use the Bluetooth to interface with the robot. Bluetooth is the most popular method to control a robot because it does not requires the robot to be limited by the length of the cable or in a direct line of sight with the controller. This project is to optimize a way for image transferring through Bluetooth while having the ability to control the movement of the robot. The programming language used is in Java Language. With the completion of this project, any Bluetooth and Java supported mobile phone will be able to control this robot and accesses the image taken from the robot. The robot will be small enough to gain access to a place where human is difficult to enter. Some problems can occur because of the limitation such as the Bluetooth bandwidth. Two mobile phone will be use to communicate with each other for data transferring. The mobile phone used will have IrDA, Bluetooth, camera and Java support on the hardware side for receiving command and transmitting image to another Bluetooth and Java supported mobile phone. Most of these mobile phones are only able to connect as slave to a PC using converter with PC act as a Master using USB cable. This communication is not available between mobile phone and normal PIC without stack and USB support. Other method will be use for communication with the robot.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

GPRS	general packet radio service
EDGE	enhanced data rates for GSM evolution
UMTS	universal mobile telecommunication system
HSPDA	high-speed downlink packet access
USB	universal serial bus
RS-232	recommended standard 232
OTG	on-the-go
PAN	personal area network
OBEX	object exchange
L2CAP	logical link control and adaptation protocol
RFCOMM	radio frequency communication
URL	uniform resource locator
JRE	Java runtime environment
MMAPI	mobile media API
CDC	connected device configuration
CLDC	connected limited device configuration
SDK	software development kit
“Server”	Mobile Phone which initiate a server for connection
‘Client”	Mobile Phone which connect to a server

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

This chapter will describes the general overview of the final year project and the contents to be included are the primary motivation and purpose, the components and device involved and the organization of this report.

1.1 Motivation for Project

The propeller-driven radio controlled boat built by Nikola Tesla in 1898, is the original prototype of all modern-day uninhabited aerial vehicles and precision guided weapon.[1] It is the beginning of all remotely controlled devices. All remotely operated devices powered by lead-acid batteries and an electric drive motor all receiving instruction from a wireless remote-control transmitter.

Nowadays, many applications have been developed using this kind of approaches for all kind of uses. Bluetooth had become mature from the past few years and it is used as the mechanism to connect more than two Bluetooth supported devices. It will not restrict the direction between devices and can be controlled in any places within the range.

A mobile phone controlled robot is defined as a robot in which react to the instruction given from a mobile phone. It does not restrict the motion but it will be limited by the distance for the Bluetooth connection. The robot will be implemented

by a camera that transfer image to the mobile phone in a desired frame per second rate. Most of the mobile phone is Java supported and the application developed for the mobile phone will be using Java programming.

A few products had been released for Bluetooth controlled robot. For example Sony Ericsson released ROB-1, a remote controlled camera that can move around and controlled by any Bluetooth enabled mobile phone[2]. Ecamm Network also introduces the World's first Bluetooth wireless webcam on January 2009[3].

Mobile phone is becoming a device that is owned by mostly everyone and it plays a very important role in our life. The current mobile phones in the market will have full colour displays, integrated cameras, fast processor and even dedicated accelerometer. According to International Data Corporation (IDC) Worldwide Quarterly Mobile Phone Tracker, the global sales of the mobile phones had grown 17.9% in third quarter 2010 as the result driven by the Smartphone sales. The Vendors shipped a total of 1.39 billion units on a cumulative worldwide basis in 2010, which is up 18.5% from the 1.17 billion units shipped in 2009[4].

Camera phones are very popular among mobile phone users which include a large range of age, and it keeps increasing. In addition of taking still images, camera phones also able to stream video using either the camera on the back for recording or the camera in the front for video phone calls. Combination of camera and the CPU of the mobile phones are capable in presenting new interaction techniques and link both physical and virtual world. With improved image resolution and computing power, camera phones can do better than just taking photo.

With the camera of mobile phones becoming common, the use of camera input for anything other than capturing still images or video will have its opportunity in developing innovative interactive applications. Mobile phones are programmable by third party developers with access to communication, computing and image capture capabilities.

1.2 Aims and Objectives

The aim of this project is to have robot that can be controlled by a mobile phone with the image "seen" by the robot projected on the mobile phone. The robot will act as the vision for mobile phone user in a place that the user cannot get to easily. The robot will represent the user and hence provides safety to the user.

This project can be divided into 3 major parts. They are user's mobile phone, interface between microcontroller with mobile phone and the microcontroller. First part will consists of the user's mobile phone; it can be any Bluetooth enabled mobile phone with Java supported. Java is used as the main programming language for the mobile phone. The objective is to develop a Java application for this mobile phone to establish Bluetooth connection and receive image from another device. It will also include the controlling for the robot. The Java programming will be the main focus in my project. The optimized way of image transferring will be developed though this part of the project.

Second part will be the interfacing between microcontroller and the mobile phone. The detail will be introduced and discussed in "MOBILE PHONE CONTROLLED ROBOT-II" report. The microcontroller will decide the movement of the robot. The microcontroller will only need to control the movement of the robot. The challenge for this part is the interface between microcontroller and mobile phone. The microcontroller has to able to establish the communication and retrieved instruction from the mobile phone.

When two Java supported mobile phone is used for image transferring, the data transfer will be easier to decode and transferred across two devices. There is an advantage when compared to transferring data from a Java programming mobile phone to C programming microcontroller.

1.3 Approaches

There are several classes of application that could be used. In this project, we will consider mainly on Mobile Media API (JSR-135) and Bluetooth (JSR-82) based Application: (1) Bluetooth Communication (such as transferring image), (2) Camera Manipulation (such as capturing image) and (3) Image Processing (such as decoding and saving image).

1.3.1 Bluetooth Communication

For communication between interfaces of mobile phone with other devices (such as another mobile phone or PC), Bluetooth is one of the method available. To use the Bluetooth as the interface method, we will need to write a Java program that will be able to initialize the Bluetooth communication using one of the Bluetooth profiles, which are definitions of possible applications and specify general behaviours. These profiles will parameterize and control the communication for the Bluetooth. Once complete, we will know how a mobile phone is able to communicate with other devices using Bluetooth as the method of communication. Some examples of J2ME Bluetooth applications for mobile phones will be shown later in this report.

1.3.2 Camera Manipulation

Developer is able to access the camera of the mobile phone with an API called Mobile Media API (MMAPI) [5]. Simple Java Programming will be created for the mobile phone to capture photo on request. In the end of the project, the mobile phone will be able to receive the command through the connection of Bluetooth. In this case, photo taken by the camera will be save in desired format and prepared to be sent trough the Bluetooth communication to another mobile phone.

1.3.3 Image Processing

The last part will be the image processing as the photo capture by the camera of mobile phone through MMAPI will not be the desired size and format for sending. Simple image processing will be developed to save the photo taken by the camera in desired format and in the right resolution size. The main purpose for this process is to help in increasing the speed of Bluetooth communication. There is more complex image processing in compressing the image size without causing the image to be unclear that involved better computational speed of current mobile phone. It can greatly increase the Bluetooth transferring rate and with better image resolution, but it is limited by the CPU speed of the mobile phone. However, the speed can also be improved by increasing the processing speed of Java program by code optimization.

1.4 Why choose Bluetooth?

1.4.1 Bluetooth vs. Infrared vs. 802.11b

Infrared is fairly reliable and its cost is low. The drawbacks of infrared are the receiver must be align with its sender and the device cannot be send to multiple receivers at the same time. The advantage is that interference is uncommon and the message is able to send into desired recipient even if there are many receivers together.

802.11b protocol is designed for large devices with lots of power and speed such as laptops and desktops. The speed is relatively higher and the distance can be larger compared to Bluetooth. Bluetooth is designed for smaller devices like PDAs or mobile phones. The range is shorter and the power requirement can be reduced.

Bluetooth does not require keeping track of cables, connectors and connection. The devices will find each other automatically and start communicating when needed. Bluetooth can also handle both data and voice and it used frequency hopping which is the spread spectrum approach that greatly prevents interception of communication. For application designed in J2ME mobile phone, it is best we choose Bluetooth as our communication.

1.5 Why use J2ME?

Mobile phones use different operating system for each other. Some of them might be using platforms and technologies such as Window Mobile[6]. Palm OS, Symbian OS, Macromedia's Flash Lite, DoCoMo's DoJa or Sun's J2ME (Java 2 Micro Edition) [9]. It is difficult to develop an application with the capability to run across all the different operating system. However, Java and J2ME are designed to provide a platform independent development tool.

Developing a Mobile Media API and Bluetooth based application that work on real time on a mobile phone requires a selection of nice techniques. The mobile platform will be limited by its computation speed and storage capacity. J2ME is one of the most common platforms for mobile application and games because of its software library that able to provide high-level user interface features such as lists and checkboxes. This characteristic enables developers to create better and user-friendly interface for their application. The applications developed under J2ME are portable between different operating systems. Therefore, J2ME application is the best way in developing mobile phone application.

1.6 Organization of Report

The report will include the use of library in Mobile Media API and Bluetooth for mobile phone and describes several mobile application based on these library. In Chapter 2, some of the camera phone applications will be reviewed. Chapter 3 gives an overview of the library structure that we used. In Chapter 4, the details of the algorithms used in building application for our project will be introduced. As for Chapter 5, experimental result and trial will be presented. Finally, conclusion for our project and some suggestion on how to improve the application in the future will be discussed in Chapter 6.

CHAPTER 2

LITERATURE REVIEW

In this chapter, existing related application available for mobile phone will be introduced and reviewed. Although many games and applications have been developed in the market, only small proportion of them is using the camera input or Bluetooth. Hence, we will first describe the existing related applications and libraries, which are developed based on the J2ME platform.

2.0 Existing J2ME Applications and Libraries

There are a few library involving mobile phone that will be used in our project, we will categorize these into 3 types of applications which described previously (Bluetooth Communication, Camera Manipulation, Image Processing).

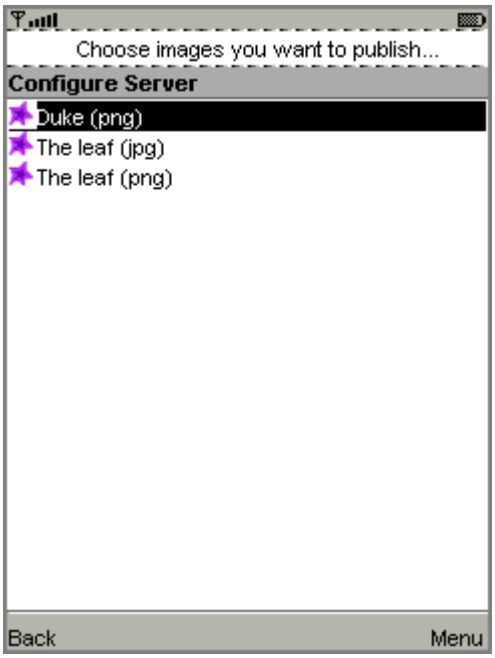
2.1 Bluetooth Communication

In application which involved Bluetooth communication, two Bluetooth enabled phone will be connected together and exchange data. Most of the multiplayer games use Bluetooth as the communication method between two or more devices. Using Bluetooth connection, two or more independent devices will be able to join together in one field and play independently on their own device. One of the applications

which demonstrate the use of Bluetooth is BluetoothDemo application by Sun Java which is included in Wireless Toolkit 2.2[12].

2.1.1 BluetoothDemo

This application contains a MIDlet that demonstrates the use of JSR-82's Bluetooth API. BluetoothDemo shows the transferring of images between two devices. The first mobile phone will be called "Server" and it is use to select the image to be transferred and Bluetooth need to be initialized at the beginning.



2.1 List to be published on Mobile Phone "Server"

On the second mobile phone, it will be called "Client" and the Bluetooth is enabled at the beginning. Pressing "Find" command will established the Bluetooth connection and searches for the image that is provided by the first mobile phone. The image will be load when the user selects the particular image on the second mobile phone.

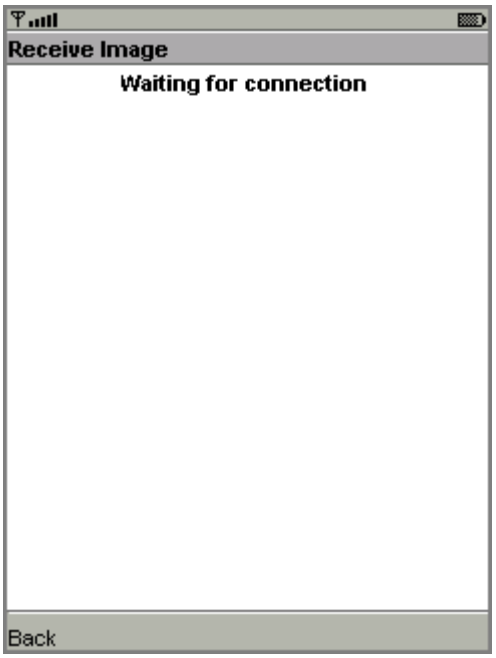
2.2 Received Image on Mobile Phone “Client”

2.2 OBEX API

Bluetooth communication is better compared to Infrared connection as Bluetooth does not require the transceiver to point directly for connection. OBEX API is one of the methods used for transferring data. ObexDemo is also an application that shows how to transfer image files between devices using the OBEX API. This demonstration is used with infrared connection. The program used in this demonstration will be used in our project which will be discussed later.

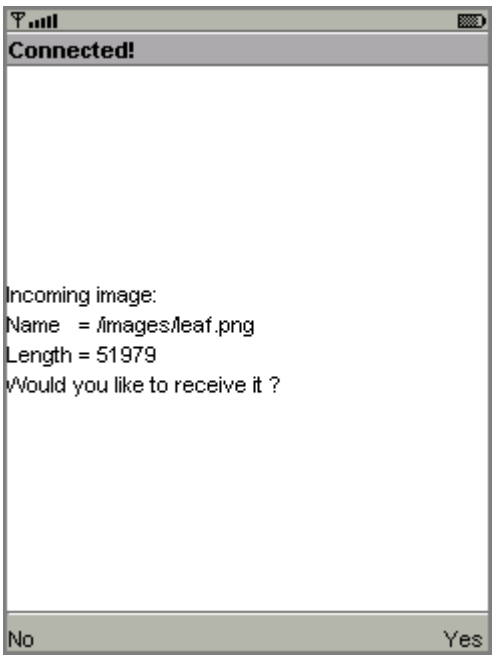
2.2.1 ObexDemo

One of the device will listens for incoming connections, while the other will attempts to send the image. The first device is set to receive image and the device will start waiting for incoming connections.



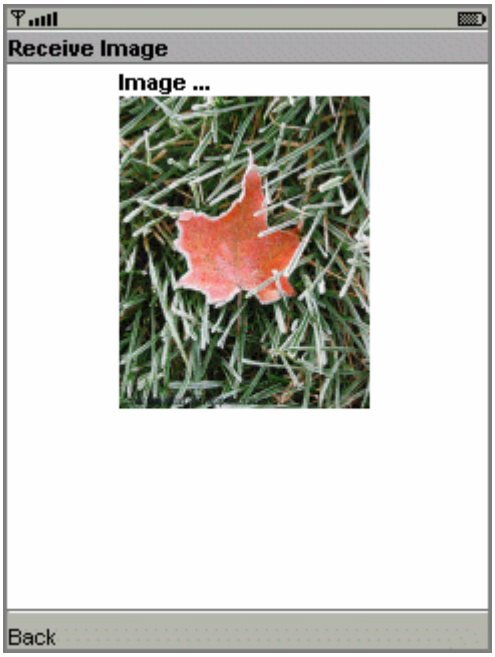
2.3 Waiting for connection using infrared

Second device will set to send image to the first device. When the first device received the connection, there will be a prompt asking whether to accept the incoming connection.



2.4 Connection established between two devices

The image sent by the second device will be displayed on the screen of the first device.



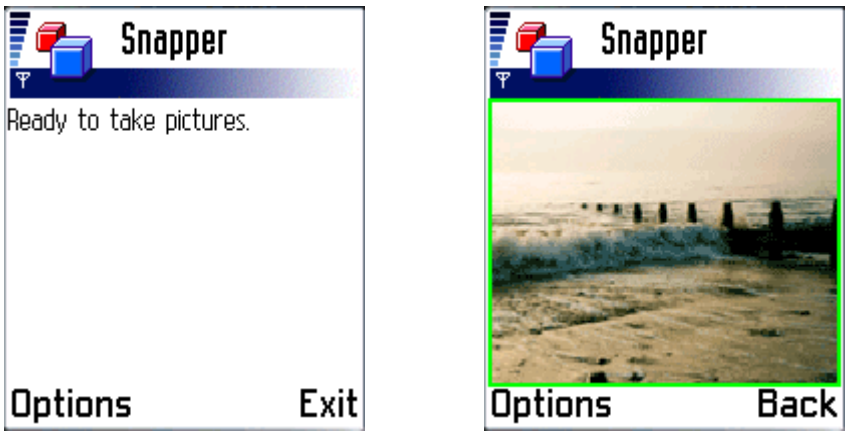
2.5 Image received through OBEX API

2.3 Camera Manipulation

To access and control the camera implemented in a mobile phone, Mobile Media API will be used. MMAPI[13] extends the functionality of the J2ME platform by providing audio,video and other time-based multimedia support to resource-constrained devices. With only a simple and easy optional package, it allows Java developers to gain permission in accessing the native multimedia services available on the device. The main purpose is based on Connected Limited Device Configuration (CLDC), but it also designed for supporting devices that based on Connected Device Configuration (CDC). One of the applications available is called Snapper.

2.3.1 Snapper

This application will be able to identify whether video capture is supported. When the camera is available, a live video coming from the camera will be displayed on the screen of the mobile phone.



2.6 Camera is initialized for image capturing

When capture button is pressed, the picture will be taken and snapper will creates a thumbnail and shows it on the MIDlet's main screen. Thumbnail image will be added to the main screen each time a picture is taken.

2.7 Thumbnail of captured image

2.4 Image Processing

In order to transfer the image faster, we need to convert the image capture into smaller size with acceptable resolution. SavingCapturedImage[14] demonstrate on how to save the image capture by the camera using MMAPI. SavingCaputuredImage is basically the same as Snapper which discussed earlier except it includes the method to save the image while Snapper is only display the thumbnail.

CHAPTER 3

Developing Application for Mobile Phone

In this Chapter, we will first introduce Java 2 Micro Edition (J2ME) platform and the tools that is used for developing the application in J2ME platform. Next, the library and APIs used will be discussed in a more detail way. In the end of the chapter, we will describe the algorithms that will be used in our application for this project.

3.1 Using Java for Mobile Phone Applications

The Chart Below shows the summary of various options for application development for mobile phones. The option will be limited by the mobile phone used in our project, which are Sony Ericsson k750i and c903i. Therefore, the best choice will be Java ME.

Platform	Overview
Java ME	Second best reach, best overall development
Flash Lite	Good for graphics-heavy applications in supported markets
Symbian	Strong support from Nokia, best access to hardware
.NET	PocketPC + Windows Mobile Devices
BREW	The only option for CDMA networks
Python	Great for quick prototypes, still immature
WAP	Largest overall reach, lightweight functionality

3.1 Summary for application development for mobile phones

There are some advantages in choosing J2ME because J2ME has the largest availability among the others by a large range and extensive developer community with lower time for re-implementation and porting. The following charts will show the relative comparisons between the different development options.

Platform	Language	X-Platform	Learning Curve	Emulator	Availability
Java ME	Java	Average	Average	Free	~1.5bn
Flash Lite	AS	Excellent	Average	With IDE	77-115m
Symbian	C++	Average	STEEP!	Free	120m
.NET	C#, C++, VB.NET	WM	STEEP!	IDE	4.5m
BREW	C++	CDMA only	STEEP!	Simulator	????
Python	Python	FREE	Gentle	Add-on	Nokia-only
WAP / Mobile Web	XHTML, WML	FREE	Gentle	Free	2bn+

3.2 Comparisons between development options

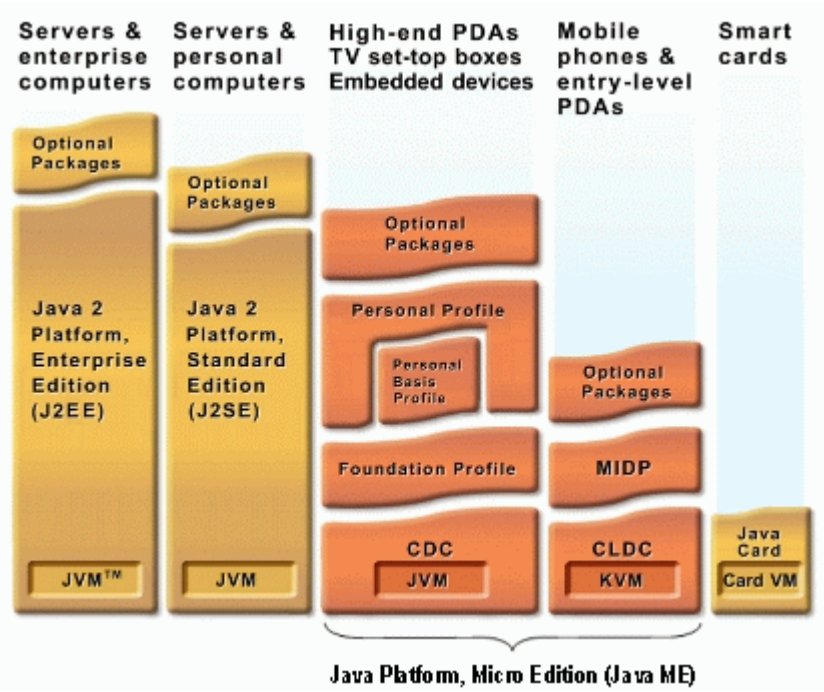
In developing a Java based image transferring application for mobile phones. There are two important aspects that must be considered:

- J2ME programming
- Connection between two devices

In this section we will review each of these elements first and then describe in detail a sample J2ME image transferring application that we have developed for a mobile phone.

3.1.1 Developing a J2ME Application

In order to run Java application on mobile device, J2ME code is necessary. J2ME is a specification of the Java platform which is focusing on providing a certified collection of Java APIs for the small, resource-constrained devices such as mobile phone, PDAs and set-top boxes.



3.2 The J2ME Platform

There are configurations, profiles, and optional API packages in J2ME. Configurations are mainly about the device memory configuration and the minimum set of APIs used for developing applications to run on a range of devices. J2ME configuration is divided into Connected Device Configuration (CDC) and Connected Limited Device Configuration (CLDC).

CDC is normally found on high-end PDAs and other devices more powerful than mobile phone. It needs a minimum requirement of 512kb of ROM and 256KB of RAMs as well as some type of network connection. CLDC is more important for us because of its support on mobile phones and other devices of small size such as pagers. CLDC requires only 160kb of ROM and 32kb of RAM, a 16-bit processor. It also has advantage such as low power consumption.

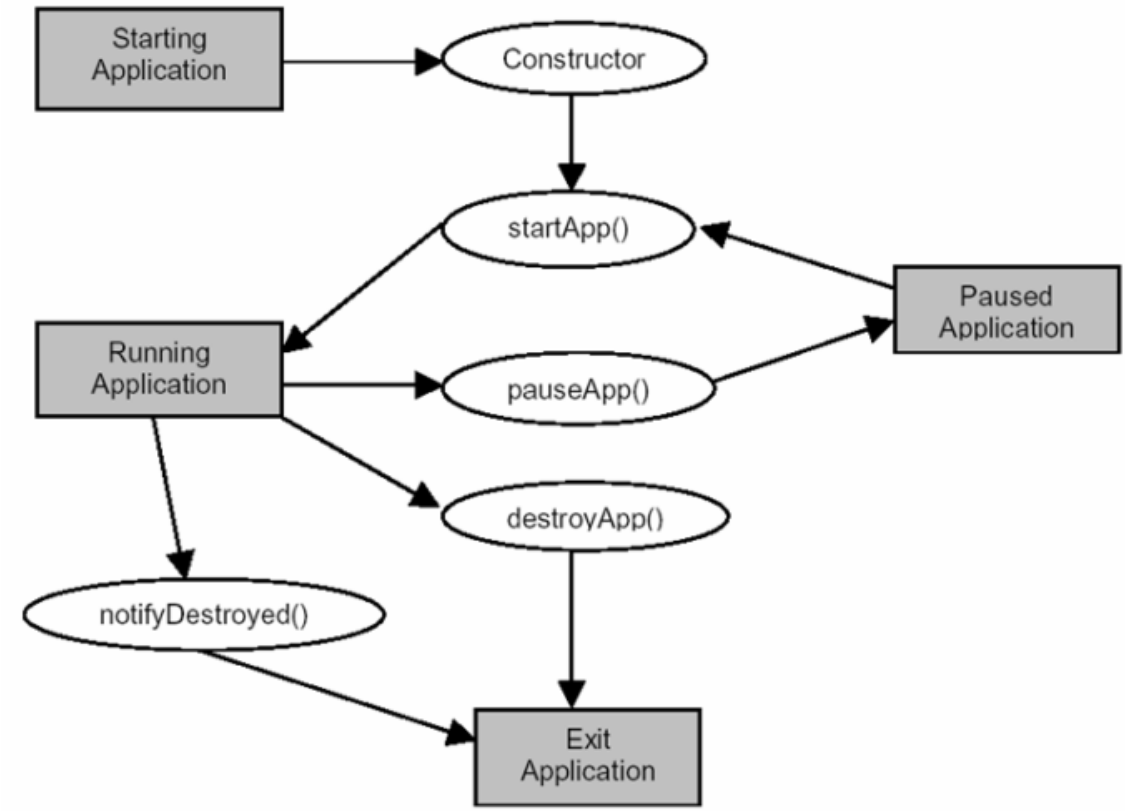
K Virtual Machine (KVM) is the reference implementation of CLDC which is based on a small JVM. The KVM only includes a subset of the bytecode verifier and it does not allow native methods to be added at runtime. There are a few versions of CLDC that have been deployed, CLDC 1.0 (JSR 30) and CLDC 1.1 (JSR 139) which added floating point data types. A profile with more specific set of APIs that further target a device is on top of a configuration. The primary focus is on Mobile Information Device Profile (MIDP) and there are MIDP 1.0 (JSR 37) and MIDP 2.0 (JSR 118). MIDP 2.0 introduces the support on multimedia (java.microedition.media), game user interface API (java.microedition.lcdui.game), and many other important features for image transferring application of mobile phones. The phones available and used on this project will need to support MIDP2.0.

MIDP hardware minimum requirements are as follows:

- 256KB of ROM for the MIDP API libraries
- 128KB of RAM for the Java runtime system
- 8KB of non-volatile writable memory for persistent application data
- Screen size of 96x54 pixels with 1-bit colour depth (black and white at least)
- Some input device, either a keypad, keyboard, or touch screen
- Two-way network of some type (intermittent is expected)

Some optional API packages include functionality that will only be supported on certain devices. Some of the APIs is found in Chapter 2 such as Java APIs for Bluetooth (JSR 82) and Mobile Media API (JSR 135). Other APIs are Java binding for OpenGL ES (JSR 239), and Advanced Graphics and User Interface (JSR 209). There are some APIs which provide access to particular features and functionality.

MIDP applications are normally called MIDlets, the diagram below shows the lifecycle of a MIDlet.



3.4 Lifecycle of a MIDlet

J2ME is a reduced version of J2SE and therefore the footprint is smaller and it does not contain classes like swing and awt. The User Interface is based on a succession of screens and it is not subsets of AWT/Swing. MIDP provides only limited UI elements which are Form, Alert, Choice and ChoiceGroup, List, StringItem, TextBox, TextField, DateField, Guage and Ticker. Some simple highlights for describing the J@ME programming from normal Java programming are:

- No floating point in CLDC 1.0, although CLDC 1.1 includes floating point support.
- No object finalization – mechanism by which objects can clean themselves up before garbage collection
- No reflection
- No native methods
- No user classloading
- Multithreading similar to J2SE, but no interrupt() method
- The J2ME Math class is a subset of J2SE version, more so with CLDC 1.0 since there is no floating point support
- The Runtime and System classes are greatly reduced from J2SE versions
- CLDC only includes a dozen classes from J2SE's java.util package

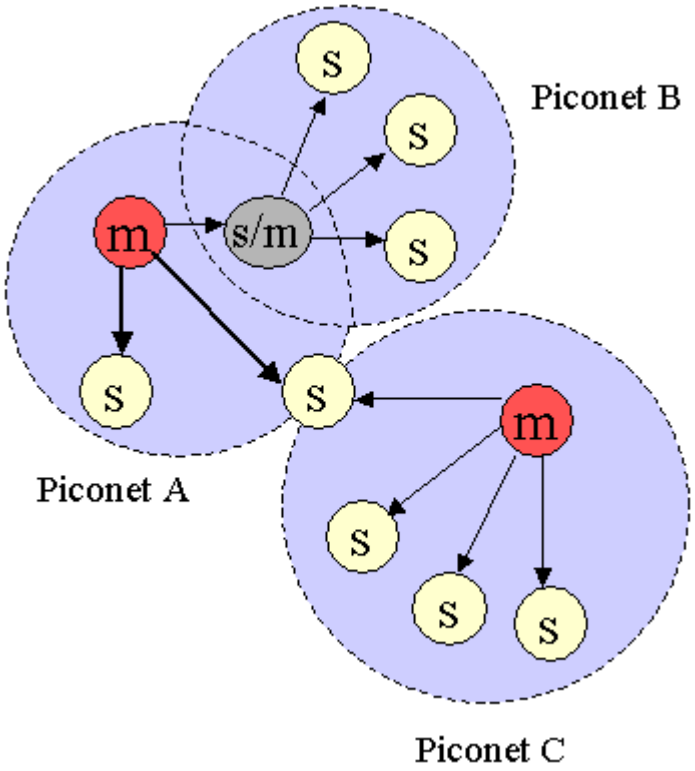
In terms of the development work for this report, the IDE used was:

- Java 2 Platform, Standard Edition
- Eclipse SDK 3.1
- EclipseME 1.0
- J2ME Wireless Toolkit (J2ME WTK) 2.2

3.2 Developing with Bluetooth Connection

3.2.1 Bluetooth Network Topology

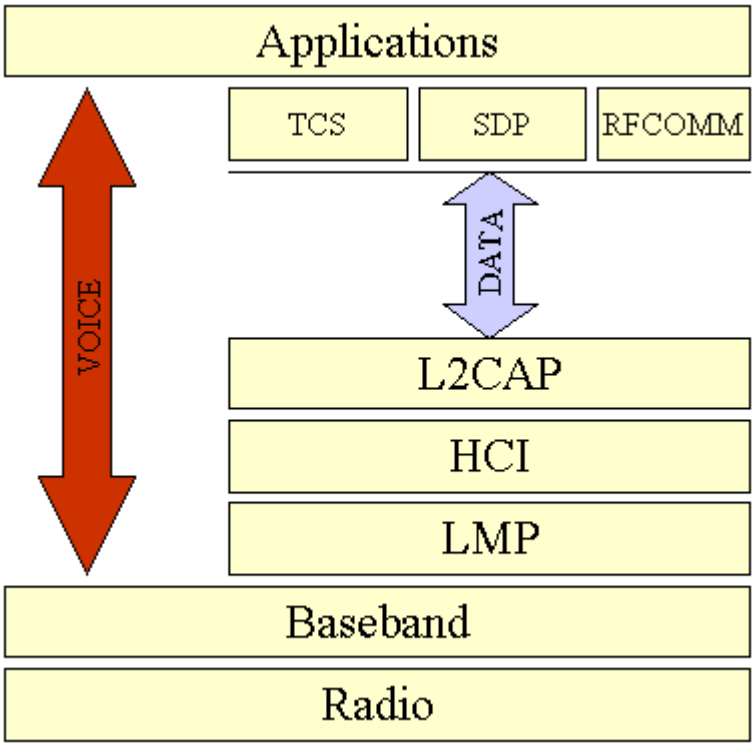
Bluetooth-enabled devices are organized in groups called piconets. One piconet consists of a master and up to seven active slaves. A master in one piconet can be a slave in another piconet.



3.5 Piconets

3.2.2 The Bluetooth Protocol Stack

The Bluetooth specification is more than 1500 pages long and it contains the information necessary. The following is the high-level view of the architecture of the Bluetooth protocol stack:



3.6 Architecture of Bluetooth Protocol Stack

Radio layer is the layer of physical wireless connection. The modulation is based on fast frequency hopping to avoid interference with other devices that communicate in the ISM band. There are 79 channels with 1 MHz apart which is divided from the 2.4GNz frequency band of Bluetooth (from 2.402 to 2.480 GHz). Bluetooth uses this spread spectrum to hop from one another with up to 1600 times a second.

Baseband layer is to control and send the data packets over radio link. It provides transmission channels for both data and voice. Baseband layer maintains Synchronous Connection-Oriented (SCO) links for voice and Asynchronous Connectionless (ACL) links for data.

Link Manager Protocol (LMP) used the links set by the baseband to establish connection and manage piconets. Their responsibilities also include authentication and security services, and monitoring of service quality.

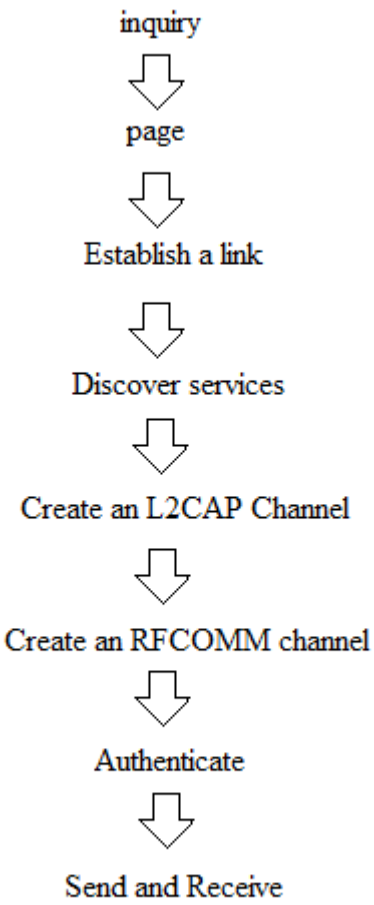
Host Controller Interface (HCI) is the driver interface for the physical bus that connects these two components. The HCI may not be required as the L2CAP can be accessed directly by the application.

Logical Link Control and Adaptation Protocol (L2CAP) receive application data and adapt it to the Bluetooth format. Quality of Service (QoS) parameter is exchanged at this layer.

3.2.3 Establishing Network Connection

A device will be in standby mode when it is not connected to a piconet. The device will listens for messages every 1.28 seconds over 32 hop frequencies. It will send our 16 identical page message on 16 hop frequencies when one device wanted to establish a connection with another. If the slave does not respond to the page message, the master must precede the page message with an inquiry message.

The steps to establish a Bluetooth connection is as the example below:



3.7 Steps for Bluetooth Connection

3.2.4 Bluetooth Profiles

Bluetooth profiles are needed to ensure the devices and applications from different manufacturers and vendors are able to operate with each other. The profile defines the role and capabilities for specific types of applications and Bluetooth devices cannot interact with other device if they have no profile.

Below are some examples of the Bluetooth profiles:

- Generic Access Profile
- Service Discovery Application and Profile
- Serial Port Profile
- LAN Access Profile
- Synchronization

The Generic Access profile defines connection procedures, device discovery, and link management. Besides that, it also defines the procedures related to the use of different security models and common format requirements for parameters accessible on the user interface level. All Bluetooth devices must support this profile as the minimum requirement.

Service Discovery Application and Profile defines the features and procedure for an application in the Bluetooth device to discover services registered in other Bluetooth devices, and retrieves information related to the services.

Serial Port Profile defines the requirements for Bluetooth devices that need to set up the connections that emulate serial cables and use the RFCOMM protocol.

LAN Access Profile defines how Bluetooth devices can access the services of a LAN using PPP, and shows how PPP mechanism can be used to form a network consisting of Bluetooth devices.

Synchronization Profile defines the application requirement for Bluetooth devices that need to synchronize data on two or more devices.

3.2.5 Bluetooth Security

Bluetooth connection provides security in three ways, pseudo-random frequency hopping, authentication, and encryption. Frequency hops make it difficult to be intersected. Authentication will allow the user to limit the connectivity to specified devices. Encryption used secret keys to make data intelligible only to authorized parties.

All Bluetooth-enabled devices must implement the generic Access Profile that contains all the Bluetooth protocols and possible devices. This profile defined three security modes:

Mode 1 is an insecure mode of operation as there is no security procedures initiated. Mode 2 is known as service-level enforced security. Devices operating in this mode will not have security procedures until the channel is established. This mode will also enable application to have different access policies and run them in parallel. Mode 3 is also known as link-level enforced security. Security procedures will be initiated before the link setup is complete.

CHAPTER 4

METHODOLOGY

In this chapter, we focus on the algorithms used in our image transferring application.

4.1 Establishing Bluetooth Communication

First we need to develop the connection for Bluetooth communication. The anatomy of Bluetooth application has five parts and they are stack initialization, device management, device discovery, service discovery, and communication.

4.1.1 Stack Initialization

In order to control the Bluetooth device, the Bluetooth stack that is responsible is needed to be initialized. For J2ME programming, the Bluetooth specification (JSR 82) is prepared and ready to be used anytime. Hence the initialization will be used directly.

4.1.2 Device Management

Two classes are introduced in this Java Bluetooth APIs, LocalDevice and RemoteDevice. LocalDevice is from javax.bluetooth.DeviceClass class and it is used to retrieve the device's type and the kinds of services it offers. RemoteDevice class provides methods for a device within the range to retrieve information about the Bluetooth address and name of the device. The following code snippet is used.

```
// create/get a local device
localDevice= LocalDevice.getLocalDevice();
```

4.1.3 Device Discovery

Wireless devices gain access to each other by using a mechanism which allows them to find each other. The core Bluetooth API's DiscoveryAgent class and DiscoveryListener interface provide the discovery services.

The DiscoveryAgent.startInquiry method will places the device into an inquiry mode and DiscoveryListener.deviceDiscovered is called each time an inquiry finds a device. DisvcoveryListener.inquiryCompleted is executed when the inquiry is completed or cancelled. The code snippet is as follows:

```
// retrieve the discovery agent
DiscoveryAgent agent = local.getDiscoveryAgent();
```

4.1.4 Service Discovery

When at least one remote device is discovered, the local device will begin to search for available services. DiscoverAgent also provides methods to discover services on Bluetooth device, and initiate service-discovery transaction.

4.1.5 Service Registration

The service of a device is needed to be registered on the Bluetooth server first before it can be discovered. The server will be responsible for:

- Creating a service record that describes the service offered
- Adding the service record to the server's Service Discovery DataBase (SDDB), so that it is visible and available to potential clients
- Registering the Bluetooth security measures associated with the service (enforced for connections with clients)
- Accepting connection for clients
- Updating the service record in the SDDB whenever the service's attributes change
- Removing or disabling the service record in the SDDB when the service is no longer available

To create new service record, invoke Connector.open with a server connection URL argument, and cast the result to a StreamConnectionNotifier.

```
StreamConnectionNotifier service =
(StreamConnectionNotifier) Connector.open("someURL");
```

To obtain the service record created by server device:

```
ServiceRecord sr = local.getRecord(service);
```

To prepare the service for client connection:

```
StreamConnection connection =
(StreamConnection) service.acceptAndOpen();
```

The service.close() is used when the server is ready to exit, close the connection and remove the service record.

4.1.6 Communication

Java APIs for Bluetooth provide mechanisms that use RFCOMM, L2CAP or OBEX as its protocol to allow connections to any services between local device and remote device. OBEX protocol is implemented independently in the core Bluetooth API because it can be used for many different transmission media such as wired, infrared or Bluetooth radio.

In our application we will use the RFCOMM protocol, which is the layer over L2CAP protocol and it emulates and RS-232 serial connection. The Serial Port Profile (SPP) will provide a stream-based interface to the RFCOMM protocol. The limitations of this protocol are:

- Two devices can share only one RFCOMM session at a time
- Up to 60 logical serial connection can be multiplexed over this session
- Single device can have at most 30 active RFCOMM services
- A device can support only one client connection at a time

4.1.7 Serial Port Profile Communication Steps

I. Construct a URL that indicates how to connect to the service and store it in the service record

II. Make the service record available to the client

III. Accept a connection from the client

IV. Send and receive data to and from the client

```
// assuming the service UID has been retrieved
String serviceURL =
    "btspp://localhost:"+serviceUID.toString());
// more explicitly:
String ServiceURL =
    "btspp://localhost:10203040607040A1B1C1DE100;name=SPP
        Server1";
try {
    // create a server connection
    StreamConnectionNotifier notifier =
       (StreamConnectionNotifier)
Connector.open(serviceURL);
    // accept client connections
    StreamConnection connection =
notifier.acceptAndOpen();
    // prepare to send/receive data
    byte buffer[] = new byte[100];
    String msg = "hello there, client";
    InputStream is = connection.openInputStream();
    OutputStream os = connection.openOutputStream();
    // send data to the client
    os.write(msg.getBytes());
    // read data from client
    is.read(buffer);
    connection.close();
} catch(IOException e) {
  e.printStackTrace();
}
```

The following code is to set up the RFCOMM connection to a server by the clients

I. Initiate a service discovery to retrieve the service record

II. Construct a connection URL using the service record

III. Open a connection to the server

IV. Send and receive data to and from the server

```
// (assuming we have the service record)
// use record to retrieve a connection URL
String url =
    record.getConnectionURL(
        record.NOAUTHENTICATE_NOENCRYPT, false);
// open a connection to the server
StreamConnection connection =
    (StreamConnection) Connector.open(url);
// Send/receive data
try {
    byte buffer[] = new byte[100];
    String msg = "hello there, server";
    InputStream is = connection.openInputStream();
    OutputStream os = connection.openOutputStream();
    // send data to the server
    os.write(msg.getBytes);
    // read data from the server
    is.read(buffer);
    connection.close();
} catch(IOException e) {
  e.printStackTrace();
}
```

At this point, the Bluetooth connection is established and it is ready to share the information.

4.2 Using Camera in J2ME

Next we need to discuss the processing of the mobile phone camera input using methods in J2ME. Mobile Media API (MMAPI) will be used in order to use the camera function in J2ME. “The MMAPI extends the functionality of the J2ME platform by providing audio, video and other time-based multimedia support to resource-constrained devices.

As a simple and lightweight optional package, it allows Java developers to gain access to native multimedia services available on a given device.”[42]. MMAPI will create its Player for the camera using its locator “capture://video”. The application can use the VideoControl to display a viewfinder on the screen and use getsnapshot function to capture the photo. The default format for the image captured will be PNG and it might also allow other supported format to be selected.

```
private VideoControl vc;
player = Manager.createPlayer("capture://video");
player.realize();
vc = (VideoControl)player.getControl("VideoControl");
vc.initDisplayMode(vc.USE_DIRECT_VIDEO, this);
vc.setVisible(true);
vc.setDisplaySize(128, 128);
player.start();
```

The code above is showing how the camera of a phone is utilized and output is displayed on the screen. Camera is initialized with Manager.createPlayer (“capture://video”) and it implements the PlayeListener interface. The player will be ready to receive call backs by using the realize() method. The initDisplayMode() method is used to initialize the video mode that shows the video in desired way.

There are two predefined values for its argument, USE_GUI_PRIMITIVE and USE_DIRECT_VIDEO. The USE_DIRECT VIDEO mode will be used in our application. The video will be directly rendered onto the canvas with this mode. The location of the video rendered can be set by the setDisplayLocation() method and it

is (0,0) by default. The size and the location of the video with respect to the canvas where it displayed can be set bu the VideoControl. In order for the video to be visible, we need to call setVisible(true). Finally the video is render by using the start() method.

4.3 Image Processing

The image saved by the method above will be used directly by transferring the image captured to the function of image transferring. The image transferring function is using byte[] as the transferring format and the image captured is also in byte[] format. Hence, the image captured can be passed to transferring function and be ready to be sent.

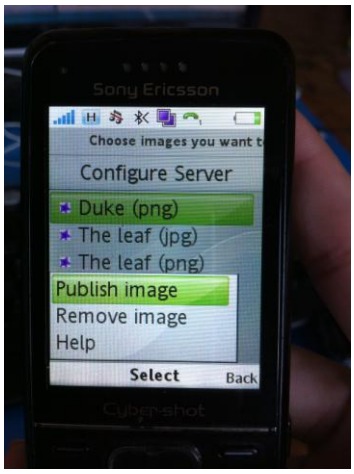
CHAPTER 5

RESULTS AND DISCUSSIONS

In this chapter, several applications and experiments that used for our algorithms will be discussed.

5.1 Bluetooth Connection

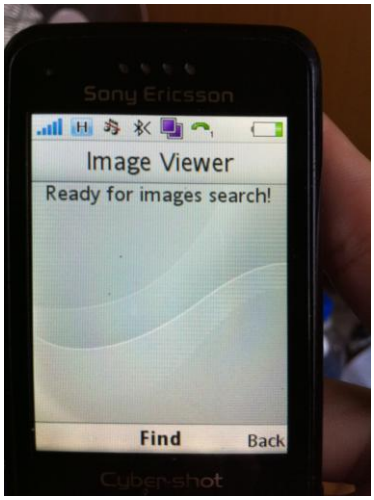
Existing application “bluetoothDemo” will be used to establish the Bluetooth connection between two mobile phones using Java programming.



5.1 Publish image to be transfer on “Server”

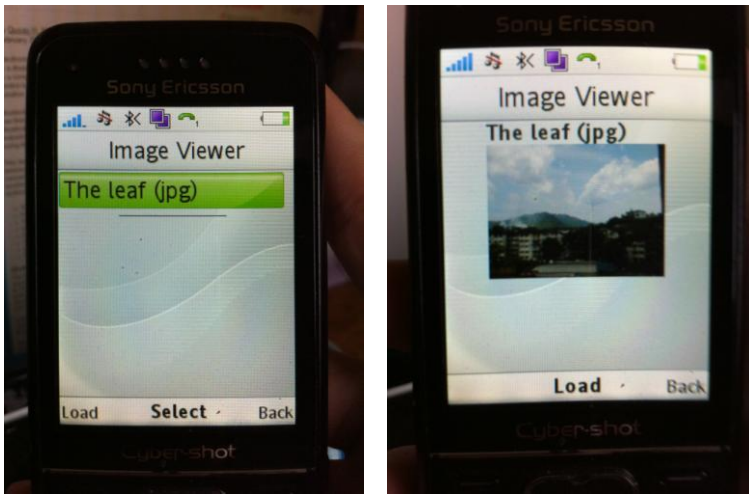
The Bluetooth Connection will be tested using two mobile phones, a “Server” and a “Client”. In this experiment, “The leaf (jpg)” will be used as example. Moreover the image is replaced with a photo taken using the camera earlier.

On the “client” mobile phone, after Bluetooth connection established, the image can be load. Hence the first part for Bluetooth connection will be finish.



5.2 Initialize Bluetooth for “Client”

When the list is loaded, the image can be loaded and the experiment on Bluetooth connection is completed. In addition, the application is modified to allow a loop of load. Hence, the image can be load again if press the “load” button. This is to allow different photo to load when the photo is updated in “Server”.



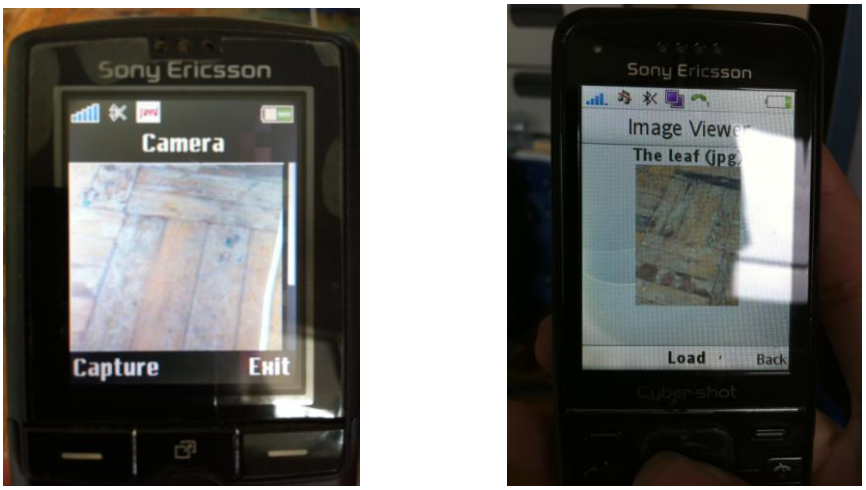
5.3 Load Published Image on “Client”

The image loaded is taken earlier and store in the package of JAR created. This application is taking the photo store in the “res” folder in the package. It cannot be used for photo taken by camera directly thus another approaches need to be achieved.

5.2 Camera Manipulation

In this part of the experiment, application “Snapper” will be used. The Application is very straight forward and it can be used directly for the application without any modification. The only challenge is to combine this application with the application developed.

After transferred the functions into the application developed, the camera can be access after the Bluetooth connection is established. For initialization, the photo in “res” folder will be used before any photo is taken.



5.4 Image capture on “Server” and Image loaded on “Client”

After a photo is taken on “Server”, the photo loaded on “Client” will be that photo. At this stage, the whole process is not automatic. Some problems occur at this stage, the first problem is the time for initialization of camera which is not constant.

5.5 White Blank Screen when Start Camera

Before the camera is in its “Ready” state, the screen will be white and blank. If the photo taking is set to be automatically, white blank screen will be sent. The problem shall be solved by adding a delay but by adding delay will cause the simulation of real time image transferring to be unsuccessful.

CHAPTER 6

CONCLUSION AND RECOMMENDATIONS

6.1 Development and Implementation

In this research, we focused on the design, development and implementation of image transferring algorithms with Bluetooth connection using J2ME for mobile phone applications. The importance and the complexity of this project are discussed in this report and the main functions and structure of the J2ME programming that we used in our algorithm is introduced. Finally, some experiment is done to develop our application.

The experiments show that one of the biggest problems in mobile phone on the J2ME platform is the fact that practically the hardware is different from the emulator provided. Some issue is not easy to be found as it is not in simulation. Although most of the mobile phones on the market are supporting Java, but the APIs, configuration and profile are different for each mobile phone and it is hard to determine the problem and improve the performance.

6.2 Technology and Problems

As the technology is move onto more advanced Smartphone, the development of J2ME platform might need replaced with the API used in Smartphone. If we can familiarize ourselves with the API either in Java or other platform, we can develop

our application in a better way. Smartphone are more powerful than mobile phones which we used in our research.

Two mobile phones is used in our project with one of the mobile phone is part of the robot because the camera that was planned to be used with the microcontroller on robot is difficult to find. Most of the components are now using USB port. So in order to interface camera with microcontroller, we need to add the USB support which is the stack from the operating system and it is almost impossible without any supplier.

6.3 Future Improvement

The project is completed without having an automatically image capturing robot and the controlling of robot is done without using Java programming. The ideal product should be a robot which controlled by the mobile phone and at the same time receiving the real time image. Hence, future improvement should be achieving an automatically image transferring

REFERENCES

[1] Christopher E. (2007). *The Robot Bat of Nikola Tesla*. Retrieved at 28 July 2010 from http://naval-history.suite101.com/article.cfm/the_robot_boat_of_nikola_tesla

[2] Darren. (2005). *Sony Ericsson ROB-1 Bluetooth Controlled Camera*. Retrieved at 28 July 2010 from http://www.livingroom.org.au/photolog/news/sony_ericsson_rob1_bluetooth_controlled_camera.php

[3] Christopher E. (2007). *The Robot Bat of Nikola Tesla*. Retrieved at 28 July 2010 from http://naval-history.suite101.com/article.cfm/the_robot_boat_of_nikola_tesla

[4] Mobilesyrup. (2011). *IDC: Worldwide Mible Phone Market*. Retrieved at 28 February 2011 from http://mobilesyrup.com/2011/01/28/idc-worldwide-mobile-phone-market-increased-17-9-in-q4-and-up-18-5-for-the-year/

[5] Oracle. (2011). *Mobile Media API (JSR 135)*. Retrieved at 2 march 2011 from http://java.sun.com/products/mmapi/

[6] Chaisatien, P. and K. Akahori (2007). "*A Pilot Study on 3G Mobile Phone and Two Dimension Barcode in Classroom Communication and Support System*". Advanced Learning Technologies, 2007. ICALT 2007. Seventh IEEE International Conference on. on Advanced Learning Technologies.

[7] Föckler, P., Zeidler, T., Brombach, B., Bruns, E., and Bimber, O. PhoneGuide: Museum Guidance Supported by On-Device Object Recognition on Mobile Phones In proceedings of International Conference on Mobile and Ubiquitous Computing (MUM'05), 2005

[8] Pulli, K., T. Aarnio, et al. (2005). "*Designing graphics programming interfaces for mobile devices.*" Computer Graphics and Applications, IEEE 25(6): 66-75.

[9] Li, S., Knudsen, J., "Beginning *J2ME: From Novice to Professional*" Apress; 3rd edition, 2005.

[10] Thompson, T. J., Kline, P. J., Kumar, C. B., & Kumar, C. B. (2008). *Bluetooth application programming with the Java APIs*. Amsterdam: Morgan Kaufmann.

[11] Wikipedia. (2007). *Java (programming language)*. Retrieved at 29 July 2010 from http://en.wikipedia.org/wiki/Java_(programming_language)

[12] Simmons, J. (2007). *Java programming*. Delhi: Global Media.

[13] Jonathan Knudsen (2003). *Taking Pictures with MMAPI*. Retrieved at 29 July 2010 from Sun Developer Network (SDN) website: http://developers.sun.com/mobility/midp/articles/picture/

APPENDICES

APPENDIX A: MIDlet (J2ME)

```
package example.bluetooth.demo;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;
import javax.microedition.midlet.MIDlet;


/**
 * Contains the Bluetooth API demo, that allows to download
 * the specific images from the other devices.
 *
 * @version ,
 */
public final class DemoMIDlet extends MIDlet implements CommandListener {
    /** The messages are shown in this demo this amount of time. */
    static final int ALERT_TIMEOUT = 2000;

    /** A list of menu items */
    private static final String[] elements = { "Server", "Client" };

    /** Soft button for exiting the demo. */
    private final Command EXIT_CMD = new Command("Exit", Command.EXIT, 2);

    /** Soft button for launching a client or sever. */
    private final Command OK_CMD = new Command("Ok", Command.SCREEN, 1);

    /** A menu list instance */
    private final List menu = new List("Bluetooth Demo", List.IMPLICIT, elements, null);
    /** A GUI part of server that publishes images. */
    private GUIImageServer imageServer;

    /** A GUI part of client that receives image from client */
    private GUIImageClient imageClient;

private CaptureAndSaveImage CASI;

    /** value is true after creating the server/client */
```

```
    private boolean isInit = false;

    /**
     * Constructs main screen of the MIDlet.
     */
    public DemoMIDlet() {
        menu.addCommand(EXIT_CMD);
        menu.addCommand(OK_CMD);
        menu.setCommandListener(this);
    }

    /**
     * Creates the demo view and action buttons.
     */
    public void startApp() {
        if (!isInit) {
            show();
        }
    }

    /**
     * Destroys the application.
     */
    protected void destroyApp(boolean unconditional) {
        if (imageServer != null) {
            imageServer.destroy();
        }

        if (imageClient != null) {
            imageClient.destroy();
        }
    }

    /**
     * Does nothing. Redefinition is required by MIDlet class.
     */
    protected void pauseApp() {
    }

    /**
     * Responds to commands issued on "client or server" form.
     *
     * @param c command object source of action
     * @param d screen object containing the item action was performed on
     */
    public void commandAction(Command c, Displayable d) {
        if (c == EXIT_CMD) {
            destroyApp(true);
            notifyDestroyed();

            return;
        }

        switch (menu.getSelectedIndex()) {
        case 0:
            imageServer = new GUIImageServer(this);

            break;
```

```
        case 1:
            //imageClient = new GUIImageClient(this);
                CASI = new CaptureAndSaveImage(this);
            break;

        default:
            System.err.println("Unexpected choice...");

            break;
        }

        isInit = true;
    }

    /** Shows main menu of MIDlet on the screen. */
    void show() {
        Display.getDisplay(this).setCurrent(menu);
    }

    /**
     * Returns the displayable object of this screen -
     * it is required for Alert construction for the error
     * cases.
     */
    Displayable getDisplayable() {
        return menu;
    }
} // end of class 'DemoMIDlet' definition
```

APPENDIX B: Bluetooth Service Searcher (J2ME)

```
package example.bluetooth.demo;

import java.io.IOException;
import java.util.Hashtable;
import java.util.Vector;

import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.UUID;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.io.StreamConnectionNotifier;

/**
 * Facade for JSR82, connecting via the <code>btspp</code> protocol.
 * This class simplifies searching for services on a device that
 * can only search for one device at the time.
 * It is based on the BlueToothFacade class found in the BlueGammon game.
 */
public class BTServiceSearcher implements DiscoveryListener
{
  /** Protocol */
  public static final String BT_PROTOCOL = "btspp";
  /** service id / server instance map */
  protected Hashtable m_servers = new Hashtable();
  /** Device Discovery lock */
  protected final Object DEVICE_LOCK = new Object();
  /** Service Discovery lock */
  protected final Object SERVICE_LOCK = new Object();
  /** Device lookup result */
  protected Vector m_devices = new Vector();
  /** Service lookup result */
  protected ServiceRecord m_record = null;



  public ServiceRecord findServiceOnDevice(String serviceNumber,
RemoteDevice device, int[] attr)
       throws IOException
       {
       ServiceRecord record = null;
    synchronized(SERVICE_LOCK)
    {
      m_record = null;
      UUID[] filter = { new UUID(serviceNumber, false) };

      DiscoveryAgent discoveryAgent =
        LocalDevice.getLocalDevice().getDiscoveryAgent();
      int trans =
        discoveryAgent.searchServices(attr, filter, device, this);

         try
      {
```

```
          SERVICE_LOCK.wait();
      } catch (InterruptedException e) {}
      record = m_record;
      m_record = null;

      return record;
    }
       }

  public ServiceRecord findServiceOnDevice(String serviceNumber,
RemoteDevice device)
       throws IOException
  {
       return findServiceOnDevice(serviceNumber, device, null);
  }

  /**
   * Setups a server if needed and returns a client. This method blocks
until a
   * client is connected. If multiple clients are allowed to be connected,
   * simply call this method multiple times.
   *
   * @param serviceNumber  The ID for the provided service
   * @return A streamconnection
   * @throws IOException
   */
  private StreamConnection waitForClient(String serviceNumber)
       throws IOException
  {
    // Set BT device to general discoverable mode
    LocalDevice.getLocalDevice().setDiscoverable(DiscoveryAgent.GIAC);

    // Accept a client
    StreamConnectionNotifier server =
      (StreamConnectionNotifier) m_servers.get(serviceNumber);
    if (server == null)
    {
      server = (StreamConnectionNotifier)Connector.open(
          BT_PROTOCOL + "://localhost:" + serviceNumber);
      m_servers.put(serviceNumber, server);
    }
    StreamConnection clientConnection = server.acceptAndOpen();
    return clientConnection;
  }

  /**
   * Closes the server setup for specified service ID.
   *
   * @param serviceNumber   The ID for the provided service
   * @throws IOException
   */
  private void closeServer(String serviceNumber) throws IOException
  {
    StreamConnectionNotifier server =
      (StreamConnectionNotifier) m_servers.get(serviceNumber);
    if (server != null)
    {
      server.close();
      m_servers.remove(serviceNumber);
      server = null;
    }
  }

  // See interface javadoc
  public void servicesDiscovered(int transID, ServiceRecord[] records)
  {
    for (int i = 0; i < records.length; i++)
    {
```

```
      String conURL =
        records[i].getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT,
false);
      if (conURL.startsWith(BT_PROTOCOL))
      {
        synchronized (SERVICE_LOCK)
        {
          m_record = records[i];
        }
        break;
      }
    }
  }

  // See interface javadoc
  public void serviceSearchCompleted(int transID, int respCode)
  {
    synchronized (SERVICE_LOCK)
    {
      SERVICE_LOCK.notifyAll();
    }
  }

  // See interface javadoc
  public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod)
  {
    synchronized(DEVICE_LOCK)
    {
      m_devices.addElement(btDevice);
    }
  }

  // See interface javadoc
  public void inquiryCompleted(int discType)
  {
    synchronized(DEVICE_LOCK)
    {
      DEVICE_LOCK.notifyAll();
    }
  }
}
```

APPENDIX C: Bluetooth Image Server (J2ME)

```
package example.bluetooth.demo;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import java.util.Hashtable;
import java.util.Vector;

// jsr082 API
import javax.bluetooth.DataElement;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.ServiceRegistrationException;
import javax.bluetooth.UUID;

// midp/cldc API
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.io.StreamConnectionNotifier;

import javax.microedition.lcdui.*;
import javax.microedition.media.*;
import javax.microedition.media.control.*;
import javax.microedition.midlet.MIDlet;


/**
 * Established the BT service, accepts connections
 * and send the requested image silently.
 *
 * @version ,
 */
final class BTImageServer implements Runnable {
    /** Describes this server */
    private static final UUID PICTURES_SERVER_UUID =
        new UUID("F0E0D0C0B0A000908070605040302010", false);

    /** The attribute id of the record item with images names. */
    private static final int IMAGES_NAMES_ATTRIBUTE_ID = 0x4321;

    /** Keeps the local device reference. */
    private LocalDevice localDevice;

    /** Accepts new connections. */
    private StreamConnectionNotifier notifier;
```

```
    /** Keeps the information about this server. */
    private ServiceRecord record;

    /** Keeps the parent reference to process specific actions. */
    private GUIImageServer parent;

    /** Becomes 'true' when this component is finalized. */
    private boolean isClosed;

    /** Creates notifier and accepts clients to be processed. */
    private Thread accepterThread;

    /** Process the particular client from queue. */
    private ClientProcessor processor;

    /** Optimization: keeps the table of data elements to be published. */
    private final Hashtable dataElements = new Hashtable();

    private Display display;

    // Form where camera viewfinder is placed
    private Form cameraForm;
    private Form thumbForm;

    // Command for capturing image by camera and saving it.
    // Placed in cameraForm.
    private Command cmdCapture = new Command("Capture", Command.OK, 0);

    // Command for exiting from midlet. Placed in cameraForm.
    private Command cmdExit = new Command("Exit", Command.EXIT, 0);

    // Player for camera
    private Player player;
    // Video control of camera
    private VideoControl videoControl;

    // Alert to be displayed if error occurs.
    private Alert alert;


private byte[] buff;
    /**
     * Constructs the bluetooth server, but it is initialized
     * in the different thread to "avoid dead lock".
     */
    BTImageServer(GUIImageServer parent) {
        this.parent = parent;

        // we have to initialize a system in different thread...
        accepterThread = new Thread(this);
        accepterThread.start();
    }

    /**
     * Accepts a new client and send him/her a requested image.
     */
    public void run() {
        boolean isBTReady = false;
```

```
        try {
            // create/get a local device
            localDevice = LocalDevice.getLocalDevice();

            // set we are discoverable
            if (!localDevice.setDiscoverable(DiscoveryAgent.GIAC)) {
                // Some implementations always return false, even if
                // setDiscoverable successful
                // throw new IOException("Can't set discoverable mode...");
            }

            // prepare a URL to create a notifier
            StringBuffer url = new StringBuffer("btspp://");

            // indicate this is a server
            url.append("localhost").append(':');

            // add the UUID to identify this service
            url.append(PICTURES_SERVER_UUID.toString());

            // add the name for our service
            url.append(";name=Picture Server");

            // request all of the client not to be authorized
            // some devices fail on authorize=true
            url.append(";authorize=false");

            // create notifier now
            notifier = (StreamConnectionNotifier)Connector.open(url.toString());

            // and remember the service record for the later updates
            record = localDevice.getRecord(notifier);

            // create a special attribute with images names
            DataElement base = new DataElement(DataElement.DATSEQ);
            record.setAttributeValue(IMAGES_NAMES_ATTRIBUTE_ID, base);

            // remember we've reached this point.
            isBTReady = true;
        } catch (Exception e) {
            System.err.println("Can't initialize bluetooth: " + e);
        }

        parent.completeInitialization(isBTReady);

        // nothing to do if no bluetooth available
        if (!isBTReady) {
            return;
        }

        // ok, start processor now
        processor = new ClientProcessor();

        // ok, start accepting connections then
        while (!isClosed) {
            StreamConnection conn = null;

            try {
                conn = notifier.acceptAndOpen();
```

```
            } catch (IOException e) {
                // wrong client or interrupted - continue anyway
                continue;
            }

            processor.addConnection(conn);
        }
    }

    /**
     * Updates the service record with the information
     * about the published images availability.
     * <p>
     * This method is invoked after the caller has checked
     * already that the real action should be done.
     *
     * @return true if record was updated successfully, false otherwise.
     */
    boolean changeImageInfo(String name, boolean isPublished) {
        // ok, get the record from service
        DataElement base = record.getAttributeValue(IMAGES_NAMES_ATTRIBUTE_ID);

        // check the corresponding DataElement object is created already
        DataElement de = (DataElement)dataElements.get(name);

        // if no, then create a new DataElement that describes this image
        if (de == null) {
            de = new DataElement(DataElement.STRING, name);
            dataElements.put(name, de);
        }

        // we know this data element has DATSEQ type
        if (isPublished) {
            base.addElement(de);
        } else {
            if (!base.removeElement(de)) {
                System.err.println("Error: item was not removed for: " + name);

                return false;
            }
        }

        record.setAttributeValue(IMAGES_NAMES_ATTRIBUTE_ID, base);

        try {
            localDevice.updateRecord(record);
        } catch (ServiceRegistrationException e) {
            System.err.println("Can't update record now for: " + name);

            return false;
        }

        return true;
    }

    /**
     * Destroy a work with bluetooth - exits the accepting
     * thread and close notifier.
     */
```

```
    void destroy() {
        isClosed = true;

        // finalize notifier work
        if (notifier != null) {
            try {
                notifier.close();
            } catch (IOException e) {
            } // ignore
        }

        // wait for acceptor thread is done
        try {
            accepterThread.join();
        } catch (InterruptedException e) {
        } // ignore

        // finalize processor
        if (processor != null) {
            processor.destroy(true);
        }

        processor = null;
    }

    /**
     * Reads the image name from the specified connection
     * and sends this image through this connection, then
     * close it after all.
     */
    private void processConnection(StreamConnection conn) {
      byte[] imgData;

        // read the image name first
        String imgName = readImageName(conn);

        // check this image is published and get the image file name
        imgName = "/images/leaf.jpg";// parent.getImageFileName(imgName);


        if(parent.raw==null) {
                // load image data into buffer to be send
        imgData = getImageData(imgName);                }
        else imgData = parent.raw;

        // send image data now
        sendImageData(imgData, conn);

        // close connection and good-bye
        try {
            conn.close();
        } catch (IOException e) {
        } // ignore
    }

private void showAlert(String title, String message, Displayable nextDisp) {
        alert = new Alert(title);
        alert.setString(message);
        alert.setTimeout(Alert.FOREVER);
```

```
        if(nextDisp != null) {
            display.setCurrent(alert, nextDisp);
        } else {
            display.setCurrent(alert);
        }
    }

    /** Send image data. */
    private void sendImageData(byte[] imgData, StreamConnection conn) {
        if (imgData == null) {
            return;
        }

        OutputStream out = null;

        try {
            out = conn.openOutputStream();
            out.write(imgData.length >> 8);
            out.write(imgData.length & 0xff);
            out.write(imgData);
            out.flush();
        } catch (IOException e) {
            System.err.println("Can't send image data: " + e);
        }

        // close output stream anyway
        if (out != null) {
            try {
                out.close();
            } catch (IOException e) {
            } // ignore
        }
    }

/**
     * Creates camera control and places it to cameraForm.
     * @throws IOException if creation of player is failed.
     * @throws MediaException if creation of player is failed.
     */
    private void createCamera() throws IOException, MediaException {
        player = Manager.createPlayer("capture://video");
        player.realize();
        player.prefetch();

        videoControl = (VideoControl)player.getControl("VideoControl");
    }

    /**
     * Adds created camera as item to cameraForm.
     */
    private void addCameraToForm() {
        cameraForm.append((Item)videoControl.
                initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null));
    }

    /**
     * Start camera player
     * @throws IOException if starting of player is failed.
```

```
     * @throws MediaException if starting of player is failed.
     */
    private void startCamera() throws IOException, MediaException  {
        if(player.getState() == Player.PREFETCHED) {
            player.start();
        }
    }

    private Image createThumbnail(Image image) {
    int sourceWidth = image.getWidth();
    int sourceHeight = image.getHeight();

    int thumbWidth = 64;
    int thumbHeight = -1;

    if (thumbHeight == -1)
      thumbHeight = thumbWidth * sourceHeight / sourceWidth;

    Image thumb = Image.createImage(thumbWidth, thumbHeight);
    Graphics g = thumb.getGraphics();

    for (int y = 0; y < thumbHeight; y++) {
      for (int x = 0; x < thumbWidth; x++) {
        g.setClip(x, y, 1, 1);
        int dx = x * sourceWidth / thumbWidth;
        int dy = y * sourceHeight / thumbHeight;
        g.drawImage(image, x - dx, y - dy, Graphics.LEFT | Graphics.TOP);
      }
    }

    Image immutableThumb = Image.createImage(thumb);

    return immutableThumb;
  }


    /** Reads image name from specified connection. */
    private String readImageName(StreamConnection conn) {
        String imgName = null;
        InputStream in = null;

        try {
            in = conn.openInputStream();

            int length = in.read(); // 'name' length is 1 byte

            if (length <= 0) {
                throw new IOException("Can't read name length");
            }

            byte[] nameData = new byte[length];
            length = 0;

            while (length != nameData.length) {
                int n = in.read(nameData, length, nameData.length - length);

                if (n == -1) {
                    throw new IOException("Can't read name data");
                }
```

```
            length += n;
        }

        imgName = new String(nameData);
    } catch (IOException e) {
        System.err.println(e);
    }

    // close input stream anyway
    if (in != null) {
        try {
            in.close();
        } catch (IOException e) {
        } // ignore
    }

    return imgName;
}

/** Reads images data from MIDlet archive to array. */
private byte[] getImageData(String imgName) {
    if (imgName == null) {
        return null;
    }

    InputStream in = getClass().getResourceAsStream(imgName);

    // read image data and create a byte array
    byte[] buff = new byte[1024];
    ByteArrayOutputStream baos = new ByteArrayOutputStream(1024);

    try {
        while (true) {
            int length = in.read(buff);

            if (length == -1) {
                break;
            }

            baos.write(buff, 0, length);
        }
    } catch (IOException e) {
        System.err.println("Can't get image data: imgName=" + imgName + " :" + e);

        return null;
    }

    return baos.toByteArray();
}

/**
 * Organizes the queue of clients to be processed,
 * processes the clients one by one until destroyed.
 */
private class ClientProcessor implements Runnable {
    private Thread processorThread;
    private Vector queue = new Vector();
    private boolean isOk = true;
```

```
ClientProcessor() {
    processorThread = new Thread(this);
    processorThread.start();
}

public void run() {



    while (!isClosed) {
        // wait for new task to be processed
        synchronized (this) {
            if (queue.size() == 0) {
                try {
                    wait();
                } catch (InterruptedException e) {
                    System.err.println("Unexpected exception: " + e);
                    destroy(false);

                    return;
                }
            }
        }

        // send the image to specified connection
        StreamConnection conn;

        synchronized (this) {
            // may be awaked by "destroy" method.
            if (isClosed) {
                return;
            }

            conn = (StreamConnection)queue.firstElement();
            queue.removeElementAt(0);
            processConnection(conn);
        }
    }
}

/** Adds the connection to queue and notifies the thread. */
void addConnection(StreamConnection conn) {
    synchronized (this) {
        queue.addElement(conn);
        notify();
    }
}

/** Closes the connections and . */
void destroy(boolean needJoin) {
    StreamConnection conn;

    synchronized (this) {
        notify();

        while (queue.size() != 0) {
            conn = (StreamConnection)queue.firstElement();
            queue.removeElementAt(0);
```

```
                    try {
                        conn.close();
                    } catch (IOException e) {
                    } // ignore
                }
            }

            // wait until dispatching thread is done
            try {
                processorThread.join();
            } catch (InterruptedException e) {
            } // ignore
        }
    }
} // end of class 'BTImageServer' definition
```

APPENDIX D: GUI Image Server (J2ME)

```
package example.bluetooth.demo;

import java.io.IOException;

import java.util.Vector;

import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.Ticker;

import javax.microedition.lcdui.*;
import javax.microedition.media.*;
import javax.microedition.media.control.*;
import javax.microedition.midlet.MIDlet;

/**
 * Allows to customize the images list to be published,
 * creates the corresponding service record to describe this list
 * and send the images to clients by request.
 *
 * @version ,
 */
final class GUIImageServer implements CommandListener {
    /** Keeps the help message of this demo. */
    private final String helpText =
        "The server is started by default.\n\n" +
        "No images are published initially. Change this by corresponding" +
        " commands - the changes have an effect immediately.\n\n" +
        "If image is removed from the published list, it can't " + "be downloaded.";

    /** This command goes to demo main screen. */
    private final Command backCommand = new Command("Back", Command.BACK, 2);

    /** Adds the selected image to the published list. */
    private final Command addCommand = new Command("Publish image", Command.SCREEN, 1);

    /** Removes the selected image from the published list. */
    private final Command removeCommand = new Command("Remove image", Command.SCREEN, 1);

    /** Shows the help message. */
    private final Command helpCommand = new Command("Help", Command.HELP, 1);

    /** The list control to configure images. */
```

```
    private final List imagesList = new List("Configure Server", List.IMPLICIT);

    /** The help screen for the server. */
    private final Alert helpScreen = new Alert("Help");

    /** Keeps the parent MIDlet reference to process specific actions. */
    private DemoMIDlet parent;

    /** The list of images file names. */
    private Vector imagesNames;

    /** These images are used to indicate the picture is published. */
    private Image onImage;

    /** These images are used to indicate the picture is published. */
    private Image offImage;

    /** Keeps an information about what images are published. */
    private boolean[] published;

    /** This object handles the real transmission. */
    private BTImageServer bt_server;

    private Display display;

    // Form where camera viewfinder is placed
    private Form cameraForm;
    private Form thumbForm;

    // Command for capturing image by camera and saving it.
    // Placed in cameraForm.
    private Command cmdCapture = new Command("Capture", Command.OK, 0);

    // Command for exiting from midlet. Placed in cameraForm.
    private Command cmdExit = new Command("Exit", Command.EXIT, 0);

    // Player for camera
    private Player player;
    // Video control of camera
    private VideoControl videoControl;

    // Alert to be displayed if error occurs.
    private Alert alert;

        public byte[] raw;


    /** Constructs images server GUI. */
    GUIImageServer(DemoMIDlet parent) {
        this.parent = parent;
        bt_server = new BTImageServer(this);
        setupIdicatorImage();
        setupImageList();
        published = new boolean[imagesList.size()];

        // prepare main screen
        imagesList.addCommand(backCommand);
        imagesList.addCommand(addCommand);
        imagesList.addCommand(removeCommand);
```

```
        imagesList.addCommand(helpCommand);
        imagesList.setCommandListener(this);

        // prepare help screen
        helpScreen.addCommand(backCommand);
        helpScreen.setTimeout(Alert.FOREVER);
        helpScreen.setString(helpText);
        helpScreen.setCommandListener(this);

        // Create camera form
        cameraForm = new Form("Camera");
           cameraForm.addCommand(cmdCapture);
        cameraForm.addCommand(cmdExit);
        cameraForm.setCommandListener(this);

        // Create thumbnail form
        thumbForm = new Form("thumb");
        thumbForm .addCommand(helpCommand);
        thumbForm .setCommandListener(this);

    }

    /**
     * Process the command event.
     *
     * @param c - the issued command.
     * @param d - the screen object the command was issued for.
     */
    public void commandAction(Command c, Displayable d) {
        if ((c == backCommand) && (d == imagesList)) {
            destroy();
            parent.show();

            return;
        }

        if ((c == backCommand) && (d == helpScreen)) {
            Display.getDisplay(parent).setCurrent(imagesList);

            return;
        }

        if (c == helpCommand) {
            //Display.getDisplay(parent).setCurrent(helpScreen);

                Display.getDisplay(parent).setCurrent(cameraForm);
                try {
            createCamera();
            addCameraToForm();
            startCamera();
        } catch(IOException ioExc) {
            showAlert("IO error", ioExc.getMessage(), null);
        } catch(MediaException mediaExc) {
            showAlert("Media error", mediaExc.getMessage(), null);
        }


                display = Display.getDisplay(parent);
        }
```

```
        if (c == cmdExit) {
                if(player != null) {
            player.deallocate();
            player.close();
        }
                Display.getDisplay(parent).setCurrent(imagesList);
        }


        if (c == cmdCapture) {
        capture();
        }
        /*
         * Changing the state of base of published images
         */
        int index = imagesList.getSelectedIndex();

        // nothing to do
        if ((c == addCommand) == published[index]) {
            return;
        }

        // update information and view
        published[index] = c == addCommand;

        Image stateImg = (c == addCommand) ? onImage : offImage;
        imagesList.set(index, imagesList.getString(index), stateImg);

        // update bluetooth service information
        if (!bt_server.changeImageInfo(imagesList.getString(index), published[index])) {
            // either a bad record or SDDB is busy
            Alert al = new Alert("Error", "Can't update base", null, AlertType.ERROR);
            al.setTimeout(DemoMIDlet.ALERT_TIMEOUT);
            Display.getDisplay(parent).setCurrent(al, imagesList);

            // restore internal information
            published[index] = !published[index];
            stateImg = published[index] ? onImage : offImage;
            imagesList.set(index, imagesList.getString(index), stateImg);
        }
    }
private void showAlert(String title, String message, Displayable nextDisp) {
        alert = new Alert(title);
        alert.setString(message);
        alert.setTimeout(Alert.FOREVER);

        if(nextDisp != null) {
            display.setCurrent(alert, nextDisp);
        } else {
            display.setCurrent(alert);
            alert.setCommandListener(this);
        }
    }

/**
     * Creates camera control and places it to cameraForm.
     * @throws IOException if creation of player is failed.
     * @throws MediaException if creation of player is failed.
```

```
     */
    private void createCamera() throws IOException, MediaException {
        player = Manager.createPlayer("capture://video");
        player.realize();
        player.prefetch();

        videoControl = (VideoControl)player.getControl("VideoControl");
    }

    /**
     * Adds created camera as item to cameraForm.
     */
    private void addCameraToForm() {
        cameraForm.append((Item)videoControl.
                initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null));
    }

    /**
     * Start camera player
     * @throws IOException if starting of player is failed.
     * @throws MediaException if starting of player is failed.
     */
    private void startCamera() throws IOException, MediaException  {
        if(player.getState() == Player.PREFETCHED) {
            player.start();
        }
    }

        public void capture() {
    try {
      // Get the image.
      raw = videoControl.getSnapshot("encoding=jpeg");
      Image image = Image.createImage(raw, 0, raw.length);

      Image thumb = createThumbnail(image);


      // Shut down the player.
      player.close();
      player = null;
      videoControl = null;

        // Place it in the main form.
        cameraForm.delete(0);
      thumbForm.deleteAll();
      thumbForm.append(thumb);

      // Flip back to the main form.
      Display.getDisplay(parent).setCurrent(thumbForm);

    }
    catch (MediaException me) { }
}

    private Image createThumbnail(Image image) {
    int sourceWidth = image.getWidth();
    int sourceHeight = image.getHeight();

    int thumbWidth = 64;
```

```
    int thumbHeight = -1;

    if (thumbHeight == -1)
      thumbHeight = thumbWidth * sourceHeight / sourceWidth;

    Image thumb = Image.createImage(thumbWidth, thumbHeight);
    Graphics g = thumb.getGraphics();

    for (int y = 0; y < thumbHeight; y++) {
      for (int x = 0; x < thumbWidth; x++) {
        g.setClip(x, y, 1, 1);
        int dx = x * sourceWidth / thumbWidth;
        int dy = y * sourceHeight / thumbHeight;
        g.drawImage(image, x - dx, y - dy, Graphics.LEFT | Graphics.TOP);
      }
    }

    Image immutableThumb = Image.createImage(thumb);

    return immutableThumb;
  }



    /**
     * We have to provide this method due to "do not do network
     * operation in command listener method" restriction, which
     * is caused by crooked midp design.
     *
     * This method is called by BTImageServer after it is done
     * with bluetooth initialization and next screen is ready
     * to appear.
     */
    void completeInitialization(boolean isBTReady) {
        // bluetooth was initialized successfully.
        if (isBTReady) {
            Ticker t = new Ticker("Choose images you want to publish...");
            imagesList.setTicker(t);
            Display.getDisplay(parent).setCurrent(imagesList);

            return;
        }

        // something wrong
        Alert al = new Alert("Error", "Can't initialize bluetooth", null, AlertType.ERROR);
        al.setTimeout(DemoMIDlet.ALERT_TIMEOUT);
        Display.getDisplay(parent).setCurrent(al, parent.getDisplayable());
    }

    /** Destroys this component. */
    void destroy() {
        // finalize the image server work
        bt_server.destroy();
    }

    /** Gets the image file name from its title (label). */
    String getImageFileName(String imgName) {
        if (imgName == null) {
            return null;
```

```
        }

        // no interface in List to get the index - should find
        int index = -1;

        for (int i = 0; i < imagesList.size(); i++) {
            if (imagesList.getString(i).equals(imgName)) {
                index = i;

                break;
            }
        }

        // not found or not published
        if ((index == -1) || !published[index]) {
            return null;
        }

        return (String)imagesNames.elementAt(index);
    }


    /**
     * Creates the image to indicate the base state.
     */
    private void setupIdicatorImage() {
        // create "on" image
        try {
            onImage = Image.createImage("/images/st-on.png");
        } catch (IOException e) {
            // provide off-screen image then
            onImage = createIndicatorImage(12, 12, 0, 255, 0);
        }

        // create "off" image
        try {
            offImage = Image.createImage("/images/st-off.png");
        } catch (IOException e) {
            // provide off-screen image then
            offImage = createIndicatorImage(12, 12, 255, 0, 0);
        }
    }

    /**
     * Gets the description of images from manifest and
     * prepares the list to control the configuration.
     * <p>
     * The attributes are named "ImageTitle-n" and "ImageImage-n".
     * The value "n" must start at "1" and be incremented by 1.
     */

    private void setupImageList() {
        imagesNames = new Vector();
        imagesList.setCommandListener(this);

        for (int n = 1; n < 100; n++) {
            String name = parent.getAppProperty("ImageName-" + n);

            // no more images available
```

```
            if ((name == null) || (name.length() == 0)) {
                break;
            }

            String label = parent.getAppProperty("ImageTitle-" + n);

            // no label available - use picture name instead
            if ((label == null) || (label.length() == 0)) {
                label = name;
            }

            imagesNames.addElement(name);
            imagesList.append(label, offImage);
        }
    }

    /**
     * Creates the off-screen image with specified size an color.
     */
    private Image createIndicatorImage(int w, int h, int r, int g, int b) {
        Image res = Image.createImage(w, h);
        Graphics gc = res.getGraphics();
        gc.setColor(r, g, b);
        gc.fillRect(0, 0, w, h);

        return res;
    }
} // end of class 'GUIImageServer' definition
```

APPENDIX E: Bluetooth Image Client (J2ME)

```
package example.bluetooth.demo;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Vector;

// jsr082 API
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DataElement;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.UUID;

// midp/cldc API
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.lcdui.Image;


/**
 * Initialize BT device, search for BT services,
 * presents them to user and picks his/her choice,
 * finally download the choosen image and present
 * it to user.
 *
 * @version ,
 */
final class BTImageClient implements Runnable, DiscoveryListener {
    /** Describes this server */
    private static final UUID PICTURES_SERVER_UUID =
        new UUID("F0E0D0C0B0A000908070605040302010", false);

    /** The attribute id of the record item with images names. */
    private static final int IMAGES_NAMES_ATTRIBUTE_ID = 0x4321;

    /** Shows the engine is ready to work. */
    private static final int READY = 0;

    /** Shows the engine is searching bluetooth devices. */
    private static final int DEVICE_SEARCH = 1;

    /** Shows the engine is searching bluetooth services. */
```

```
    private static final int SERVICE_SEARCH = 2;

    /** Keeps the current state of engine. */
    private int state = READY;

    /** Keeps the discovery agent reference. */
    private DiscoveryAgent discoveryAgent;

    /** Keeps the parent reference to process specific actions. */
    private GUIImageClient parent;

    /** Becomes 'true' when this component is finalized. */
    private boolean isClosed;

    /** Process the search/download requests. */
    private Thread processorThread;

    /** Collects the remote devices found during a search. */
    private Vector /* RemoteDevice */ devices = new Vector();

    /** Collects the services found during a search. */
    private Vector /* ServiceRecord */ records = new Vector();

    /** Keeps the device discovery return code. */
    private int discType;

    /** Keeps the services search IDs (just to be able to cancel them). */
    private int[] searchIDs;

    /** Keeps the image name to be load. */
    private String imageNameToLoad;

    /** Keeps the table of {name, Service} to process the user choice. */
    private Hashtable base = new Hashtable();

    /** Informs the thread the download should be canceled. */
    private boolean isDownloadCanceled;

    /** Optimization: keeps service search pattern. */
    private UUID[] uuidSet;

    /** Optimization: keeps attributes list to be retrieved. */
    private int[] attrSet;

    /**
     * Constructs the bluetooth server, but it is initialized
     * in the different thread to "avoid dead lock".
     */
    BTImageClient(GUIImageClient parent) {
        this.parent = parent;

        // we have to initialize a system in different thread...
        processorThread = new Thread(this);
        processorThread.start();
    }

    /**
     * Process the search/download requests.
     */
```

```
    public void run() {
        // initialize bluetooth first
        boolean isBTReady = false;

        try {
            // create/get a local device and discovery agent
            LocalDevice localDevice = LocalDevice.getLocalDevice();
            discoveryAgent = localDevice.getDiscoveryAgent();

            // remember we've reached this point.
            isBTReady = true;
        } catch (Exception e) {
            System.err.println("Can't initialize bluetooth: " + e);
        }

        parent.completeInitialization(isBTReady);

        // nothing to do if no bluetooth available
        if (!isBTReady) {
            return;
        }

        // initialize some optimization variables
        uuidSet = new UUID[2];

        // ok, we are interesting in btspp services only
        uuidSet[0] = new UUID(0x1101);

        // and only known ones, that allows pictures
        uuidSet[1] = PICTURES_SERVER_UUID;

        // we need an only service attribute actually
        attrSet = new int[1];

        // it's "images names" one
        attrSet[0] = IMAGES_NAMES_ATTRIBUTE_ID;

        // start processing the images search/download
        processImagesSearchDownload();
    }

    /**
     * Invoked by system when a new remote device is found -
     * remember the found device.
     */
    public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {
        // same device may found several times during single search
        if (devices.indexOf(btDevice) == -1) {
            devices.addElement(btDevice);
        }
    }

    /**
     * Invoked by system when device discovery is done.
     * <p>
     * Use a trick here - just remember the discType
     * and process its evaluation in another thread.
     */
    public void inquiryCompleted(int discType) {
```

```
        this.discType = discType;

        synchronized (this) {
            notify();
        }
    }

    public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {
        for (int i = 0; i < servRecord.length; i++) {
            records.addElement(servRecord[i]);
        }
    }

    public void serviceSearchCompleted(int transID, int respCode) {
        // first, find the service search transaction index
        int index = -1;

        for (int i = 0; i < searchIDs.length; i++) {
            if (searchIDs[i] == transID) {
                index = i;

                break;
            }
        }

        // error - unexpected transaction index
        if (index == -1) {
            System.err.println("Unexpected transaction index: " + transID);

            // FIXME: process the error case
        } else {
            searchIDs[index] = -1;
        }

        /*
         * Actually, we do not care about the response code -
         * if device is not reachable or no records, etc.
         */

        // make sure it was the last transaction
        for (int i = 0; i < searchIDs.length; i++) {
            if (searchIDs[i] != -1) {
                return;
            }
        }

        // ok, all of the transactions are completed
        synchronized (this) {
            notify();
        }
    }

    /** Sets the request to search the devices/services. */
    void requestSearch() {
        synchronized (this) {
            notify();
        }
    }
```

```
    /** Cancel's the devices/services search. */
    void cancelSearch() {
        synchronized (this) {
            if (state == DEVICE_SEARCH) {
                discoveryAgent.cancelInquiry(this);
            } else if (state == SERVICE_SEARCH) {
                for (int i = 0; i < searchIDs.length; i++) {
                    discoveryAgent.cancelServiceSearch(searchIDs[i]);
                }
            }
        }
    }

    /** Sets the request to load the specified image. */
    void requestLoad(String name) {
        synchronized (this) {
            imageNameToLoad = name;
            notify();
        }
    }

    /** Cancel's the image download. */
    void cancelLoad() {
        /*
         * The image download process is done by
         * this class's thread (not by a system one),
         * so no need to wake up the current thread -
         * it's running already.
         */
        isDownloadCanceled = true;
    }

    /**
     * Destroy a work with bluetooth - exits the accepting
     * thread and close notifier.
     */
    void destroy() {
        synchronized (this) {
            isClosed = true;
            isDownloadCanceled = true;
            notify();

            // FIXME: implement me
        }

        // wait for acceptor thread is done
        try {
            processorThread.join();
        } catch (InterruptedException e) {
        } // ignore
    }

    /**
     * Processes images search/download until component is closed
     * or system error has happen.
     */
    private synchronized void processImagesSearchDownload() {
        while (!isClosed) {
            // wait for new search request from user
```

```
            state = READY;

            try {
                wait();
            } catch (InterruptedException e) {
                System.err.println("Unexpected interruption: " + e);

                return;
            }

            // check the component is destroyed
            if (isClosed) {
                return;
            }

            // search for devices
            if (!searchDevices()) {
                return;
            } else if (devices.size() == 0) {
                continue;
            }

            // search for services now
            if (!searchServices()) {
                return;
            } else if (records.size() == 0) {
                continue;
            }

            // ok, something was found - present the result to user now
            if (!presentUserSearchResults()) {
                // services are found, but no names there
                continue;
            }

            // the several download requests may be processed
            while (true) {
                // this download is not canceled, right?
                isDownloadCanceled = false;

                // ok, wait for download or need to wait for next search
                try {
                    wait();
                } catch (InterruptedException e) {
                    System.err.println("Unexpected interruption: " + e);

                    return;
                }

                // check the component is destroyed
                if (isClosed) {
                    return;
                }

                // this means "go to the beginning"
                if (imageNameToLoad == null) {
                    break;
                }
```

```
                // load selected image data
                Image img = loadImage();

                // FIXME: this never happen - monitor is taken...
                if (isClosed) {
                    return;
                }

                if (isDownloadCanceled) {
                    continue; // may be next image to be download
                }

                if (img == null) {
                    parent.informLoadError("Can't load image: " + imageNameToLoad);

                    continue; // may be next image to be download
                }

                // ok, show image to user
                parent.showImage(img, imageNameToLoad);

                // may be next image to be download
                continue;
            }
        }
    }

    /**
     * Search for bluetooth devices.
     *
     * @return false if should end the component work.
     */
    private boolean searchDevices() {
        // ok, start a new search then
        state = DEVICE_SEARCH;
        devices.removeAllElements();

        try {
            discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
        } catch (BluetoothStateException e) {
            System.err.println("Can't start inquiry now: " + e);
            parent.informSearchError("Can't start device search");

            return true;
        }

        try {
            wait(); // until devices are found
        } catch (InterruptedException e) {
            System.err.println("Unexpected interruption: " + e);

            return false;
        }

        // this "wake up" may be caused by 'destroy' call
        if (isClosed) {
            return false;
        }
```

```
        // no?, ok, let's check the return code then
        switch (discType) {
        case INQUIRY_ERROR:
            parent.informSearchError("Device discovering error...");

        // fall through
        case INQUIRY_TERMINATED:
            // make sure no garbage in found devices list
            devices.removeAllElements();

            // nothing to report - go to next request
            break;

        case INQUIRY_COMPLETED:

            if (devices.size() == 0) {
                parent.informSearchError("No devices in range");
            }

            // go to service search now
            break;

        default:
            // what kind of system you are?... :(
            System.err.println("system error:" + " unexpected device discovery code: " +
discType);
            destroy();

            return false;
        }

        return true;
    }

    /**
     * Search for proper service.
     *
     * @return false if should end the component work.
     */
    private boolean searchServices() {
        state = SERVICE_SEARCH;
        records.removeAllElements();
        searchIDs = new int[devices.size()];

        boolean isSearchStarted = false;

        for (int i = 0; i < devices.size(); i++) {
            RemoteDevice rd = (RemoteDevice)devices.elementAt(i);

            try {
                searchIDs[i] = discoveryAgent.searchServices(attrSet, uuidSet, rd, this);
            } catch (BluetoothStateException e) {
                System.err.println("Can't search services for: " + rd.getBluetoothAddress()
+
                    " due to " + e);
                searchIDs[i] = -1;

                continue;
            }
```

```
            isSearchStarted = true;
        }

        // at least one of the services search should be found
        if (!isSearchStarted) {
            parent.informSearchError("Can't search services.");

            return true;
        }

        try {
            wait(); // until services are found
        } catch (InterruptedException e) {
            System.err.println("Unexpected interruption: " + e);

            return false;
        }

        // this "wake up" may be caused by 'destroy' call
        if (isClosed) {
            return false;
        }

        // actually, no services were found
        if (records.size() == 0) {
            parent.informSearchError("No proper services were found");
        }

        return true;
    }

    /**
     * Gets the collection of the images titles (names)
     * from the services, prepares a hashtable to match
     * the image name to a services list, presents the images names
     * to user finally.
     *
     * @return false if no names in found services.
     */
    private boolean presentUserSearchResults() {
        base.clear();

        for (int i = 0; i < records.size(); i++) {
            ServiceRecord sr = (ServiceRecord)records.elementAt(i);

            // get the attribute with images names
            DataElement de = sr.getAttributeValue(IMAGES_NAMES_ATTRIBUTE_ID);

            if (de == null) {
                System.err.println("Unexpected service - missed attribute");

                continue;
            }

            // get the images names from this attribute
            Enumeration deEnum = (Enumeration)de.getValue();

            while (deEnum.hasMoreElements()) {
```

```
                de = (DataElement)deEnum.nextElement();

                String name = (String)de.getValue();

                // name may be stored already
                Object obj = base.get(name);

                // that's either the ServiceRecord or Vector
                if (obj != null) {
                    Vector v;

                    if (obj instanceof ServiceRecord) {
                        v = new Vector();
                        v.addElement(obj);
                    } else {
                        v = (Vector)obj;
                    }

                    v.addElement(sr);
                    obj = v;
                } else {
                    obj = sr;
                }

                base.put(name, obj);
            }
        }

        return parent.showImagesNames(base);
    }

    /**
     * Loads selected image data.
     */
    private Image loadImage() {
        if (imageNameToLoad == null) {
            System.err.println("Error: imageNameToLoad=null");

            return null;
        }

        // ok, get the list of service records
        ServiceRecord[] sr = null;
        Object obj = base.get(imageNameToLoad);

        if (obj == null) {
            System.err.println("Error: no record for: " + imageNameToLoad);

            return null;
        } else if (obj instanceof ServiceRecord) {
            sr = new ServiceRecord[] { (ServiceRecord)obj };
        } else {
            Vector v = (Vector)obj;
            sr = new ServiceRecord[v.size()];

            for (int i = 0; i < v.size(); i++) {
                sr[i] = (ServiceRecord)v.elementAt(i);
            }
        }
```

```
        // now try to load the image from each services one by one
        for (int i = 0; i < sr.length; i++) {
            StreamConnection conn = null;
            String url = null;

            // the process may be canceled
            if (isDownloadCanceled) {
                return null;
            }

            // first - connect
            try {
                url = sr[i].getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT,
false);
                conn = (StreamConnection)Connector.open(url);
            } catch (IOException e) {
                System.err.println("Note: can't connect to: " + url);

                // ignore
                continue;
            }

            // then open a steam and write a name
            try {
                OutputStream out = conn.openOutputStream();
                out.write(imageNameToLoad.length()); // length is 1 byte
                out.write(imageNameToLoad.getBytes());
                out.flush();
                out.close();
            } catch (IOException e) {
                System.err.println("Can't write to server for: " + url);

                // close stream connection
                try {
                    conn.close();
                } catch (IOException ee) {
                } // ignore

                continue;
            }

            // then open a steam and read an image
            byte[] imgData = null;

            try {
                InputStream in = conn.openInputStream();

                // read a length first
                int length = in.read() << 8;
                length |= in.read();

                if (length <= 0) {
                    throw new IOException("Can't read a length");
                }

                // read the image now
                imgData = new byte[length];
                length = 0;
```

```
                while (length != imgData.length) {
                    int n = in.read(imgData, length, imgData.length - length);

                    if (n == -1) {
                        throw new IOException("Can't read a image data");
                    }

                    length += n;
                }

                in.close();
            } catch (IOException e) {
                // SEMC BEGIN:
                System.err.println("Can't read from server for: " + url + ". Message was: "
+ e);
                // SEMC END
                continue;
            } finally {
                // close stream connection anyway
                try {
                    conn.close();
                } catch (IOException e) {
                } // ignore
            }

            // ok, may it's a chance
            Image img = null;

            try {
                img = Image.createImage(imgData, 0, imgData.length);
            } catch (Exception e) {
                // may be next time
                System.err.println("Error: wrong image data from: " + url);

                continue;
            }

            return img;
        }

        return null;
    }
} // end of class 'BTImageClient' definition
```

APPENDIX F: GUI Image Client (J2ME)

```
package example.bluetooth.demo;

import java.io.IOException;
import java.util.Enumeration;
import java.util.Hashtable;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Gauge;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.ImageItem;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.StringItem;


/**
 * Provides a GUI to present the download options
 * to used, gives a chance to make a choice,
 * finally shows the downloaded image.
*/
final class GUIImageClient implements CommandListener {
    /** This command goes to demo main screen. */
    private final Command SCR_MAIN_BACK_CMD = new Command("Back", Command.BACK, 2);

    /** Starts the proper services search. */
    private final Command SCR_MAIN_SEARCH_CMD = new Command("Find", Command.OK, 1);

    /** Cancels the device/services discovering. */
    private final Command SCR_SEARCH_CANCEL_CMD = new Command("Cancel", Command.BACK, 2);

    /** This command goes to client main screen. */
    private final Command SCR_IMAGES_BACK_CMD = new Command("Back", Command.BACK, 2);

    /** Start the chosen image download. */
    private final Command SCR_IMAGES_LOAD_CMD = new Command("Load", Command.OK, 1);

    /** Cancels the image download. */
    private final Command SCR_LOAD_CANCEL_CMD = new Command("Cancel", Command.BACK, 2);

    /** This command goes from image screen to images list one. */
    private final Command SCR_SHOW_BACK_CMD = new Command("Back", Command.BACK, 2);

    /** The main screen of the client part. */
    private final Form mainScreen = new Form("Image Viewer");

    /** The screen with found images names. */
```

```
    private final List listScreen = new List("Image Viewer", List.IMPLICIT);

    /** The screen with download image. */
    private final Form imageScreen = new Form("Image Viewer");

    /** Keeps the parent MIDlet reference to process specific actions. */
    private DemoMIDlet parent;

    /** This object handles the real transmission. */
    private BTImageClient bt_client;

    /** Constructs client GUI. */
    GUIImageClient(DemoMIDlet parent) {
        this.parent = parent;
        bt_client = new BTImageClient(this);
        mainScreen.addCommand(SCR_MAIN_BACK_CMD);
        mainScreen.addCommand(SCR_MAIN_SEARCH_CMD);
        mainScreen.setCommandListener(this);
        listScreen.addCommand(SCR_IMAGES_BACK_CMD);
        listScreen.addCommand(SCR_IMAGES_LOAD_CMD);
        listScreen.setCommandListener(this);
        imageScreen.addCommand(SCR_SHOW_BACK_CMD);
        imageScreen.setCommandListener(this);
    }

    /**
     * Process the command events.
     *
     * @param c - the issued command.
     * @param d - the screen object the command was issued for.
     */
    public void commandAction(Command c, Displayable d) {
        // back to demo main screen
        if (c == SCR_MAIN_BACK_CMD) {
            destroy();
            parent.show();

            return;
        }

        // starts images (device/services) search
        if (c == SCR_MAIN_SEARCH_CMD) {
            Form f = new Form("Searching...");
            f.addCommand(SCR_SEARCH_CANCEL_CMD);
            f.setCommandListener(this);
            f.append(new Gauge("Searching images...", false, Gauge.INDEFINITE,
                    Gauge.CONTINUOUS_RUNNING));
            Display.getDisplay(parent).setCurrent(f);
            bt_client.requestSearch();

            return;
        }

        // cancels device/services search
        if (c == SCR_SEARCH_CANCEL_CMD) {
            bt_client.cancelSearch();
            Display.getDisplay(parent).setCurrent(mainScreen);

            return;
```

```
        }

        // back to client main screen
        if (c == SCR_IMAGES_BACK_CMD) {
            bt_client.requestLoad(null);
            Display.getDisplay(parent).setCurrent(mainScreen);

            return;
        }

        // starts image download
        if (c == SCR_IMAGES_LOAD_CMD) {
            Form f = new Form("Loading...");
            f.addCommand(SCR_LOAD_CANCEL_CMD);
            f.setCommandListener(this);
            f.append(new Gauge("Loading image...", false, Gauge.INDEFINITE,
Gauge.CONTINUOUS_RUNNING));
            Display.getDisplay(parent).setCurrent(f);

            List l = (List)d;
            bt_client.requestLoad(l.getString(l.getSelectedIndex()));

            return;
        }

        // cancels image load
        if (c == SCR_LOAD_CANCEL_CMD) {
            bt_client.cancelLoad();
            Display.getDisplay(parent).setCurrent(listScreen);

            return;
        }

        // back to client main screen
        if (c == SCR_SHOW_BACK_CMD) {
            Display.getDisplay(parent).setCurrent(listScreen);

            return;
        }
    }

    /**
     * We have to provide this method due to "do not do network
     * operation in command listener method" restriction, which
     * is caused by crooked midp design.
     *
     * This method is called by BTImageClient after it is done
     * with bluetooth initialization and next screen is ready
     * to appear.
     */
    void completeInitialization(boolean isBTReady) {
        // bluetooth was initialized successfully.
        if (isBTReady) {
            StringItem si = new StringItem("Ready for images search!",
null);
            si.setLayout(StringItem.LAYOUT_CENTER |
StringItem.LAYOUT_VCENTER);
            mainScreen.append(si);
            Display.getDisplay(parent).setCurrent(mainScreen);

            return;
        }

        // something wrong
        Alert al = new Alert("Error", "Can't initialize bluetooth", null,
AlertType.ERROR);
        al.setTimeout(DemoMIDlet.ALERT_TIMEOUT);
        Display.getDisplay(parent).setCurrent(al, parent.getDisplayable());
```

```
    }

    /** Destroys this component. */
    void destroy() {
        // finalize the image client work
        bt_client.destroy();
    }

    /**
     * Informs the error during the images search.
     */
    void informSearchError(String resMsg) {
        Alert al = new Alert("Error", resMsg, null, AlertType.ERROR);
        al.setTimeout(DemoMIDlet.ALERT_TIMEOUT);
        Display.getDisplay(parent).setCurrent(al, mainScreen);
    }

    /**
     * Informs the error during the selected image load.
     */
    void informLoadError(String resMsg) {
        Alert al = new Alert("Error", resMsg, null, AlertType.ERROR);
        al.setTimeout(DemoMIDlet.ALERT_TIMEOUT);
        Display.getDisplay(parent).setCurrent(al, listScreen);
    }

    /**
     * Shows the downloaded image.
     */
    void showImage(Image img, String imgName) {
        imageScreen.deleteAll();
        imageScreen.append(new ImageItem(imgName, img,
                ImageItem.LAYOUT_CENTER | ImageItem.LAYOUT_VCENTER,
"Downloaded image: " + imgName));
        Display.getDisplay(parent).setCurrent(imageScreen);
    }

    /**
     * Shows the available images names.
     * @return false if no images names were found actually
     */
    boolean showImagesNames(Hashtable base) {
        Enumeration keys = base.keys();

        // no images actually
        if (!keys.hasMoreElements()) {
            informSearchError("No images names in found services");

            return false;
        }

        // prepare the list to be shown
        while (listScreen.size() != 0) {
            listScreen.delete(0);
        }

        while (keys.hasMoreElements()) {
            listScreen.append((String)keys.nextElement(), null);
        }

        Display.getDisplay(parent).setCurrent(listScreen);

        return true;
    }
} // end of class 'GUIImageClient' definition
```

APPENDICES

APPENDIX A: MIDlet (J2ME)

```
package example.bluetooth.demo;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;
import javax.microedition.midlet.MIDlet;


/**
 * Contains the Bluetooth API demo, that allows to download
 * the specific images from the other devices.
 *
 * @version ,
 */
public final class DemoMIDlet extends MIDlet implements CommandListener {
    /** The messages are shown in this demo this amount of time. */
    static final int ALERT_TIMEOUT = 2000;

    /** A list of menu items */
    private static final String[] elements = { "Server", "Client" };

    /** Soft button for exiting the demo. */
    private final Command EXIT_CMD = new Command("Exit", Command.EXIT, 2);

    /** Soft button for launching a client or sever. */
    private final Command OK_CMD = new Command("Ok", Command.SCREEN, 1);

    /** A menu list instance */
    private final List menu = new List("Bluetooth Demo", List.IMPLICIT, elements, null);
    /** A GUI part of server that publishes images. */
    private GUIImageServer imageServer;

    /** A GUI part of client that receives image from client */
    private GUIImageClient imageClient;

private CaptureAndSaveImage CASI;

    /** value is true after creating the server/client */
    private boolean isInit = false;
```

```
    /**
     * Constructs main screen of the MIDlet.
     */
    public DemoMIDlet() {
        menu.addCommand(EXIT_CMD);
        menu.addCommand(OK_CMD);
        menu.setCommandListener(this);
    }

    /**
     * Creates the demo view and action buttons.
     */
    public void startApp() {
        if (!isInit) {
            show();
        }
    }

    /**
     * Destroys the application.
     */
    protected void destroyApp(boolean unconditional) {
        if (imageServer != null) {
            imageServer.destroy();
        }

        if (imageClient != null) {
            imageClient.destroy();
        }
    }

    /**
     * Does nothing. Redefinition is required by MIDlet class.
     */
    protected void pauseApp() {
    }

    /**
     * Responds to commands issued on "client or server" form.
     *
     * @param c command object source of action
     * @param d screen object containing the item action was performed on
     */
    public void commandAction(Command c, Displayable d) {
        if (c == EXIT_CMD) {
            destroyApp(true);
            notifyDestroyed();

            return;
        }

        switch (menu.getSelectedIndex()) {
        case 0:
            imageServer = new GUIImageServer(this);

            break;
```

```
        case 1:
            //imageClient = new GUIImageClient(this);
```

```
                CASI = new CaptureAndSaveImage(this);
            break;

        default:
            System.err.println("Unexpected choice...");

            break;
        }

        isInit = true;
    }

    /** Shows main menu of MIDlet on the screen. */
    void show() {
        Display.getDisplay(this).setCurrent(menu);
    }

    /**
     * Returns the displayable object of this screen -
     * it is required for Alert construction for the error
     * cases.
     */
    Displayable getDisplayable() {
        return menu;
    }
} // end of class 'DemoMIDlet' definition
```

APPENDIX B: Bluetooth Service Searcher (J2ME)

```
package example.bluetooth.demo;

import java.io.IOException;
import java.util.Hashtable;
import java.util.Vector;

import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.UUID;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.io.StreamConnectionNotifier;

/**
 * Facade for JSR82, connecting via the <code>btspp</code> protocol.
 * This class simplifies searching for services on a device that
 * can only search for one device at the time.
 * It is based on the BlueToothFacade class found in the BlueGammon game.
 */
public class BTServiceSearcher implements DiscoveryListener
{
  /** Protocol */
  public static final String BT_PROTOCOL = "btspp";
  /** service id / server instance map */
  protected Hashtable m_servers = new Hashtable();
  /** Device Discovery lock */
  protected final Object DEVICE_LOCK = new Object();
  /** Service Discovery lock */
  protected final Object SERVICE_LOCK = new Object();
  /** Device lookup result */
  protected Vector m_devices = new Vector();
  /** Service lookup result */
  protected ServiceRecord m_record = null;



  public ServiceRecord findServiceOnDevice(String serviceNumber,
RemoteDevice device, int[] attr)
       throws IOException
       {
       ServiceRecord record = null;
    synchronized(SERVICE_LOCK)
    {
      m_record = null;
      UUID[] filter = { new UUID(serviceNumber, false) };

      DiscoveryAgent discoveryAgent =
        LocalDevice.getLocalDevice().getDiscoveryAgent();
      int trans =
        discoveryAgent.searchServices(attr, filter, device, this);

         try
      {
```

```
          SERVICE_LOCK.wait();
      } catch (InterruptedException e) {}
      record = m_record;
      m_record = null;

      return record;
    }
       }

  public ServiceRecord findServiceOnDevice(String serviceNumber,
RemoteDevice device)
       throws IOException
  {
       return findServiceOnDevice(serviceNumber, device, null);
  }

  /**
   * Setups a server if needed and returns a client. This method blocks
until a
   * client is connected. If multiple clients are allowed to be connected,
   * simply call this method multiple times.
   *
   * @param serviceNumber  The ID for the provided service
   * @return A streamconnection
   * @throws IOException
   */
  private StreamConnection waitForClient(String serviceNumber)
       throws IOException
  {
    // Set BT device to general discoverable mode
    LocalDevice.getLocalDevice().setDiscoverable(DiscoveryAgent.GIAC);

    // Accept a client
    StreamConnectionNotifier server =
      (StreamConnectionNotifier) m_servers.get(serviceNumber);
    if (server == null)
    {
      server = (StreamConnectionNotifier)Connector.open(
          BT_PROTOCOL + "://localhost:" + serviceNumber);
      m_servers.put(serviceNumber, server);
    }
    StreamConnection clientConnection = server.acceptAndOpen();
    return clientConnection;
  }

  /**
   * Closes the server setup for specified service ID.
   *
   * @param serviceNumber   The ID for the provided service
   * @throws IOException
   */
  private void closeServer(String serviceNumber) throws IOException
  {
    StreamConnectionNotifier server =
      (StreamConnectionNotifier) m_servers.get(serviceNumber);
    if (server != null)
    {
      server.close();
      m_servers.remove(serviceNumber);
      server = null;
    }
  }

  // See interface javadoc
  public void servicesDiscovered(int transID, ServiceRecord[] records)
  {
    for (int i = 0; i < records.length; i++)
    {
```

```
      String conURL =
        records[i].getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT,
false);
      if (conURL.startsWith(BT_PROTOCOL))
      {
        synchronized (SERVICE_LOCK)
        {
          m_record = records[i];
        }
        break;
      }
    }
  }

  // See interface javadoc
  public void serviceSearchCompleted(int transID, int respCode)
  {
    synchronized (SERVICE_LOCK)
    {
      SERVICE_LOCK.notifyAll();
    }
  }

  // See interface javadoc
  public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod)
  {
    synchronized(DEVICE_LOCK)
    {
      m_devices.addElement(btDevice);
    }
  }

  // See interface javadoc
  public void inquiryCompleted(int discType)
  {
    synchronized(DEVICE_LOCK)
    {
      DEVICE_LOCK.notifyAll();
    }
  }
}
```

APPENDIX C: Bluetooth Image Server (J2ME)

```
package example.bluetooth.demo;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import java.util.Hashtable;
import java.util.Vector;

// jsr082 API
import javax.bluetooth.DataElement;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.ServiceRegistrationException;
import javax.bluetooth.UUID;

// midp/cldc API
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.io.StreamConnectionNotifier;

import javax.microedition.lcdui.*;
import javax.microedition.media.*;
import javax.microedition.media.control.*;
import javax.microedition.midlet.MIDlet;


/**
 * Established the BT service, accepts connections
 * and send the requested image silently.
 *
 * @version ,
 */
final class BTImageServer implements Runnable {
    /** Describes this server */
    private static final UUID PICTURES_SERVER_UUID =
        new UUID("F0E0D0C0B0A000908070605040302010", false);

    /** The attribute id of the record item with images names. */
    private static final int IMAGES_NAMES_ATTRIBUTE_ID = 0x4321;

    /** Keeps the local device reference. */
    private LocalDevice localDevice;

    /** Accepts new connections. */
    private StreamConnectionNotifier notifier;
```

```
    /** Keeps the information about this server. */
    private ServiceRecord record;

    /** Keeps the parent reference to process specific actions. */
    private GUIImageServer parent;

    /** Becomes 'true' when this component is finalized. */
    private boolean isClosed;

    /** Creates notifier and accepts clients to be processed. */
    private Thread accepterThread;

    /** Process the particular client from queue. */
    private ClientProcessor processor;

    /** Optimization: keeps the table of data elements to be published. */
    private final Hashtable dataElements = new Hashtable();

    private Display display;

    // Form where camera viewfinder is placed
    private Form cameraForm;
    private Form thumbForm;

    // Command for capturing image by camera and saving it.
    // Placed in cameraForm.
    private Command cmdCapture = new Command("Capture", Command.OK, 0);

    // Command for exiting from midlet. Placed in cameraForm.
    private Command cmdExit = new Command("Exit", Command.EXIT, 0);

    // Player for camera
    private Player player;
    // Video control of camera
    private VideoControl videoControl;

    // Alert to be displayed if error occurs.
    private Alert alert;


private byte[] buff;
    /**
     * Constructs the bluetooth server, but it is initialized
     * in the different thread to "avoid dead lock".
     */
    BTImageServer(GUIImageServer parent) {
        this.parent = parent;

        // we have to initialize a system in different thread...
        accepterThread = new Thread(this);
        accepterThread.start();
    }

    /**
     * Accepts a new client and send him/her a requested image.
     */
    public void run() {
        boolean isBTReady = false;
```

```
        try {
            // create/get a local device
            localDevice = LocalDevice.getLocalDevice();

            // set we are discoverable
            if (!localDevice.setDiscoverable(DiscoveryAgent.GIAC)) {
                // Some implementations always return false, even if
                // setDiscoverable successful
                // throw new IOException("Can't set discoverable mode...");
            }

            // prepare a URL to create a notifier
            StringBuffer url = new StringBuffer("btspp://");

            // indicate this is a server
            url.append("localhost").append(':');

            // add the UUID to identify this service
            url.append(PICTURES_SERVER_UUID.toString());

            // add the name for our service
            url.append(";name=Picture Server");

            // request all of the client not to be authorized
            // some devices fail on authorize=true
            url.append(";authorize=false");

            // create notifier now
            notifier = (StreamConnectionNotifier)Connector.open(url.toString());

            // and remember the service record for the later updates
            record = localDevice.getRecord(notifier);

            // create a special attribute with images names
            DataElement base = new DataElement(DataElement.DATSEQ);
            record.setAttributeValue(IMAGES_NAMES_ATTRIBUTE_ID, base);

            // remember we've reached this point.
            isBTReady = true;
        } catch (Exception e) {
            System.err.println("Can't initialize bluetooth: " + e);
        }

        parent.completeInitialization(isBTReady);

        // nothing to do if no bluetooth available
        if (!isBTReady) {
            return;
        }

        // ok, start processor now
        processor = new ClientProcessor();

        // ok, start accepting connections then
        while (!isClosed) {
            StreamConnection conn = null;

            try {
                conn = notifier.acceptAndOpen();
```

```
            } catch (IOException e) {
                // wrong client or interrupted - continue anyway
                continue;
            }

            processor.addConnection(conn);
        }
    }

    /**
     * Updates the service record with the information
     * about the published images availability.
     * <p>
     * This method is invoked after the caller has checked
     * already that the real action should be done.
     *
     * @return true if record was updated successfully, false otherwise.
     */
    boolean changeImageInfo(String name, boolean isPublished) {
        // ok, get the record from service
        DataElement base = record.getAttributeValue(IMAGES_NAMES_ATTRIBUTE_ID);

        // check the corresponding DataElement object is created already
        DataElement de = (DataElement)dataElements.get(name);

        // if no, then create a new DataElement that describes this image
        if (de == null) {
            de = new DataElement(DataElement.STRING, name);
            dataElements.put(name, de);
        }

        // we know this data element has DATSEQ type
        if (isPublished) {
            base.addElement(de);
        } else {
            if (!base.removeElement(de)) {
                System.err.println("Error: item was not removed for: " + name);

                return false;
            }
        }

        record.setAttributeValue(IMAGES_NAMES_ATTRIBUTE_ID, base);

        try {
            localDevice.updateRecord(record);
        } catch (ServiceRegistrationException e) {
            System.err.println("Can't update record now for: " + name);

            return false;
        }

        return true;
    }

    /**
     * Destroy a work with bluetooth - exits the accepting
     * thread and close notifier.
     */
```

```
    void destroy() {
        isClosed = true;

        // finalize notifier work
        if (notifier != null) {
            try {
                notifier.close();
            } catch (IOException e) {
            } // ignore
        }

        // wait for acceptor thread is done
        try {
            accepterThread.join();
        } catch (InterruptedException e) {
        } // ignore

        // finalize processor
        if (processor != null) {
            processor.destroy(true);
        }

        processor = null;
    }

    /**
     * Reads the image name from the specified connection
     * and sends this image through this connection, then
     * close it after all.
     */
    private void processConnection(StreamConnection conn) {
      byte[] imgData;

        // read the image name first
        String imgName = readImageName(conn);

        // check this image is published and get the image file name
        imgName = "/images/leaf.jpg";// parent.getImageFileName(imgName);


        if(parent.raw==null) {
                // load image data into buffer to be send
        imgData = getImageData(imgName);                }
        else imgData = parent.raw;

        // send image data now
        sendImageData(imgData, conn);

        // close connection and good-bye
        try {
            conn.close();
        } catch (IOException e) {
        } // ignore
    }

private void showAlert(String title, String message, Displayable nextDisp) {
        alert = new Alert(title);
        alert.setString(message);
        alert.setTimeout(Alert.FOREVER);
```

```
        if(nextDisp != null) {
            display.setCurrent(alert, nextDisp);
        } else {
            display.setCurrent(alert);
        }
    }

    /** Send image data. */
    private void sendImageData(byte[] imgData, StreamConnection conn) {
        if (imgData == null) {
            return;
        }

        OutputStream out = null;

        try {
            out = conn.openOutputStream();
            out.write(imgData.length >> 8);
            out.write(imgData.length & 0xff);
            out.write(imgData);
            out.flush();
        } catch (IOException e) {
            System.err.println("Can't send image data: " + e);
        }

        // close output stream anyway
        if (out != null) {
            try {
                out.close();
            } catch (IOException e) {
            } // ignore
        }
    }

/**
     * Creates camera control and places it to cameraForm.
     * @throws IOException if creation of player is failed.
     * @throws MediaException if creation of player is failed.
     */
    private void createCamera() throws IOException, MediaException {
        player = Manager.createPlayer("capture://video");
        player.realize();
        player.prefetch();

        videoControl = (VideoControl)player.getControl("VideoControl");
    }

    /**
     * Adds created camera as item to cameraForm.
     */
    private void addCameraToForm() {
        cameraForm.append((Item)videoControl.
                initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null));
    }

    /**
     * Start camera player
     * @throws IOException if starting of player is failed.
```

```
     * @throws MediaException if starting of player is failed.
     */
    private void startCamera() throws IOException, MediaException  {
        if(player.getState() == Player.PREFETCHED) {
            player.start();
        }
    }

    private Image createThumbnail(Image image) {
    int sourceWidth = image.getWidth();
    int sourceHeight = image.getHeight();

    int thumbWidth = 64;
    int thumbHeight = -1;

    if (thumbHeight == -1)
      thumbHeight = thumbWidth * sourceHeight / sourceWidth;

    Image thumb = Image.createImage(thumbWidth, thumbHeight);
    Graphics g = thumb.getGraphics();

    for (int y = 0; y < thumbHeight; y++) {
      for (int x = 0; x < thumbWidth; x++) {
        g.setClip(x, y, 1, 1);
        int dx = x * sourceWidth / thumbWidth;
        int dy = y * sourceHeight / thumbHeight;
        g.drawImage(image, x - dx, y - dy, Graphics.LEFT | Graphics.TOP);
      }
    }

    Image immutableThumb = Image.createImage(thumb);

    return immutableThumb;
  }


    /** Reads image name from specified connection. */
    private String readImageName(StreamConnection conn) {
        String imgName = null;
        InputStream in = null;

        try {
            in = conn.openInputStream();

            int length = in.read(); // 'name' length is 1 byte

            if (length <= 0) {
                throw new IOException("Can't read name length");
            }

            byte[] nameData = new byte[length];
            length = 0;

            while (length != nameData.length) {
                int n = in.read(nameData, length, nameData.length - length);

                if (n == -1) {
                    throw new IOException("Can't read name data");
                }
```

```
            length += n;
        }

        imgName = new String(nameData);
    } catch (IOException e) {
        System.err.println(e);
    }

    // close input stream anyway
    if (in != null) {
        try {
            in.close();
        } catch (IOException e) {
        } // ignore
    }

    return imgName;
}

/** Reads images data from MIDlet archive to array. */
private byte[] getImageData(String imgName) {
    if (imgName == null) {
        return null;
    }

    InputStream in = getClass().getResourceAsStream(imgName);

    // read image data and create a byte array
    byte[] buff = new byte[1024];
    ByteArrayOutputStream baos = new ByteArrayOutputStream(1024);

    try {
        while (true) {
            int length = in.read(buff);

            if (length == -1) {
                break;
            }

            baos.write(buff, 0, length);
        }
    } catch (IOException e) {
        System.err.println("Can't get image data: imgName=" + imgName + " :" + e);

        return null;
    }

    return baos.toByteArray();
}

/**
 * Organizes the queue of clients to be processed,
 * processes the clients one by one until destroyed.
 */
private class ClientProcessor implements Runnable {
    private Thread processorThread;
    private Vector queue = new Vector();
    private boolean isOk = true;
```

```
ClientProcessor() {
    processorThread = new Thread(this);
    processorThread.start();
}

public void run() {



    while (!isClosed) {
        // wait for new task to be processed
        synchronized (this) {
            if (queue.size() == 0) {
                try {
                    wait();
                } catch (InterruptedException e) {
                    System.err.println("Unexpected exception: " + e);
                    destroy(false);

                    return;
                }
            }
        }

        // send the image to specified connection
        StreamConnection conn;

        synchronized (this) {
            // may be awaked by "destroy" method.
            if (isClosed) {
                return;
            }

            conn = (StreamConnection)queue.firstElement();
            queue.removeElementAt(0);
            processConnection(conn);
        }
    }
}

/** Adds the connection to queue and notifies the thread. */
void addConnection(StreamConnection conn) {
    synchronized (this) {
        queue.addElement(conn);
        notify();
    }
}

/** Closes the connections and . */
void destroy(boolean needJoin) {
    StreamConnection conn;

    synchronized (this) {
        notify();

        while (queue.size() != 0) {
            conn = (StreamConnection)queue.firstElement();
            queue.removeElementAt(0);
```

```
                    try {
                        conn.close();
                    } catch (IOException e) {
                    } // ignore
                }
            }

            // wait until dispatching thread is done
            try {
                processorThread.join();
            } catch (InterruptedException e) {
            } // ignore
        }
    }
} // end of class 'BTImageServer' definition
```

APPENDIX D: GUI Image Server (J2ME)

```
package example.bluetooth.demo;

import java.io.IOException;

import java.util.Vector;

import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.Ticker;

import javax.microedition.lcdui.*;
import javax.microedition.media.*;
import javax.microedition.media.control.*;
import javax.microedition.midlet.MIDlet;

/**
 * Allows to customize the images list to be published,
 * creates the corresponding service record to describe this list
 * and send the images to clients by request.
 *
 * @version ,
 */
final class GUIImageServer implements CommandListener {
    /** Keeps the help message of this demo. */
    private final String helpText =
        "The server is started by default.\n\n" +
        "No images are published initially. Change this by corresponding" +
        " commands - the changes have an effect immediately.\n\n" +
        "If image is removed from the published list, it can't " + "be downloaded.";

    /** This command goes to demo main screen. */
    private final Command backCommand = new Command("Back", Command.BACK, 2);

    /** Adds the selected image to the published list. */
    private final Command addCommand = new Command("Publish image", Command.SCREEN, 1);

    /** Removes the selected image from the published list. */
    private final Command removeCommand = new Command("Remove image", Command.SCREEN, 1);

    /** Shows the help message. */
    private final Command helpCommand = new Command("Help", Command.HELP, 1);

    /** The list control to configure images. */
```

```
    private final List imagesList = new List("Configure Server", List.IMPLICIT);

    /** The help screen for the server. */
    private final Alert helpScreen = new Alert("Help");

    /** Keeps the parent MIDlet reference to process specific actions. */
    private DemoMIDlet parent;

    /** The list of images file names. */
    private Vector imagesNames;

    /** These images are used to indicate the picture is published. */
    private Image onImage;

    /** These images are used to indicate the picture is published. */
    private Image offImage;

    /** Keeps an information about what images are published. */
    private boolean[] published;

    /** This object handles the real transmission. */
    private BTImageServer bt_server;

    private Display display;

    // Form where camera viewfinder is placed
    private Form cameraForm;
    private Form thumbForm;

    // Command for capturing image by camera and saving it.
    // Placed in cameraForm.
    private Command cmdCapture = new Command("Capture", Command.OK, 0);

    // Command for exiting from midlet. Placed in cameraForm.
    private Command cmdExit = new Command("Exit", Command.EXIT, 0);

    // Player for camera
    private Player player;
    // Video control of camera
    private VideoControl videoControl;

    // Alert to be displayed if error occurs.
    private Alert alert;

        public byte[] raw;


    /** Constructs images server GUI. */
    GUIImageServer(DemoMIDlet parent) {
        this.parent = parent;
        bt_server = new BTImageServer(this);
        setupIdicatorImage();
        setupImageList();
        published = new boolean[imagesList.size()];

        // prepare main screen
        imagesList.addCommand(backCommand);
        imagesList.addCommand(addCommand);
        imagesList.addCommand(removeCommand);
```

```
        imagesList.addCommand(helpCommand);
        imagesList.setCommandListener(this);

        // prepare help screen
        helpScreen.addCommand(backCommand);
        helpScreen.setTimeout(Alert.FOREVER);
        helpScreen.setString(helpText);
        helpScreen.setCommandListener(this);

        // Create camera form
        cameraForm = new Form("Camera");
           cameraForm.addCommand(cmdCapture);
        cameraForm.addCommand(cmdExit);
        cameraForm.setCommandListener(this);

        // Create thumbnail form
        thumbForm = new Form("thumb");
        thumbForm .addCommand(helpCommand);
        thumbForm .setCommandListener(this);

    }

    /**
     * Process the command event.
     *
     * @param c - the issued command.
     * @param d - the screen object the command was issued for.
     */
    public void commandAction(Command c, Displayable d) {
        if ((c == backCommand) && (d == imagesList)) {
            destroy();
            parent.show();

            return;
        }

        if ((c == backCommand) && (d == helpScreen)) {
            Display.getDisplay(parent).setCurrent(imagesList);

            return;
        }

        if (c == helpCommand) {
            //Display.getDisplay(parent).setCurrent(helpScreen);

                Display.getDisplay(parent).setCurrent(cameraForm);
                try {
            createCamera();
            addCameraToForm();
            startCamera();
        } catch(IOException ioExc) {
            showAlert("IO error", ioExc.getMessage(), null);
        } catch(MediaException mediaExc) {
            showAlert("Media error", mediaExc.getMessage(), null);
        }


                display = Display.getDisplay(parent);
        }
```

```
        if (c == cmdExit) {
                if(player != null) {
            player.deallocate();
            player.close();
        }
                Display.getDisplay(parent).setCurrent(imagesList);
        }


        if (c == cmdCapture) {
        capture();
        }
        /*
         * Changing the state of base of published images
         */
        int index = imagesList.getSelectedIndex();

        // nothing to do
        if ((c == addCommand) == published[index]) {
            return;
        }

        // update information and view
        published[index] = c == addCommand;

        Image stateImg = (c == addCommand) ? onImage : offImage;
        imagesList.set(index, imagesList.getString(index), stateImg);

        // update bluetooth service information
        if (!bt_server.changeImageInfo(imagesList.getString(index), published[index])) {
            // either a bad record or SDDB is busy
            Alert al = new Alert("Error", "Can't update base", null, AlertType.ERROR);
            al.setTimeout(DemoMIDlet.ALERT_TIMEOUT);
            Display.getDisplay(parent).setCurrent(al, imagesList);

            // restore internal information
            published[index] = !published[index];
            stateImg = published[index] ? onImage : offImage;
            imagesList.set(index, imagesList.getString(index), stateImg);
        }
    }
private void showAlert(String title, String message, Displayable nextDisp) {
        alert = new Alert(title);
        alert.setString(message);
        alert.setTimeout(Alert.FOREVER);

        if(nextDisp != null) {
            display.setCurrent(alert, nextDisp);
        } else {
            display.setCurrent(alert);
            alert.setCommandListener(this);
        }
    }

/**
     * Creates camera control and places it to cameraForm.
     * @throws IOException if creation of player is failed.
     * @throws MediaException if creation of player is failed.
```

```
     */
    private void createCamera() throws IOException, MediaException {
        player = Manager.createPlayer("capture://video");
        player.realize();
        player.prefetch();

        videoControl = (VideoControl)player.getControl("VideoControl");
    }

    /**
     * Adds created camera as item to cameraForm.
     */
    private void addCameraToForm() {
        cameraForm.append((Item)videoControl.
                initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null));
    }

    /**
     * Start camera player
     * @throws IOException if starting of player is failed.
     * @throws MediaException if starting of player is failed.
     */
    private void startCamera() throws IOException, MediaException  {
        if(player.getState() == Player.PREFETCHED) {
            player.start();
        }
    }

        public void capture() {
    try {
      // Get the image.
      raw = videoControl.getSnapshot("encoding=jpeg");
      Image image = Image.createImage(raw, 0, raw.length);

      Image thumb = createThumbnail(image);


      // Shut down the player.
      player.close();
      player = null;
      videoControl = null;

        // Place it in the main form.
        cameraForm.delete(0);
      thumbForm.deleteAll();
      thumbForm.append(thumb);

      // Flip back to the main form.
      Display.getDisplay(parent).setCurrent(thumbForm);

    }
    catch (MediaException me) { }
  }

    private Image createThumbnail(Image image) {
    int sourceWidth = image.getWidth();
    int sourceHeight = image.getHeight();

    int thumbWidth = 64;
```

```
    int thumbHeight = -1;

    if (thumbHeight == -1)
      thumbHeight = thumbWidth * sourceHeight / sourceWidth;

    Image thumb = Image.createImage(thumbWidth, thumbHeight);
    Graphics g = thumb.getGraphics();

    for (int y = 0; y < thumbHeight; y++) {
      for (int x = 0; x < thumbWidth; x++) {
        g.setClip(x, y, 1, 1);
        int dx = x * sourceWidth / thumbWidth;
        int dy = y * sourceHeight / thumbHeight;
        g.drawImage(image, x - dx, y - dy, Graphics.LEFT | Graphics.TOP);
      }
    }

    Image immutableThumb = Image.createImage(thumb);

    return immutableThumb;
  }



    /**
     * We have to provide this method due to "do not do network
     * operation in command listener method" restriction, which
     * is caused by crooked midp design.
     *
     * This method is called by BTImageServer after it is done
     * with bluetooth initialization and next screen is ready
     * to appear.
     */
    void completeInitialization(boolean isBTReady) {
        // bluetooth was initialized successfully.
        if (isBTReady) {
            Ticker t = new Ticker("Choose images you want to publish...");
            imagesList.setTicker(t);
            Display.getDisplay(parent).setCurrent(imagesList);

            return;
        }

        // something wrong
        Alert al = new Alert("Error", "Can't initialize bluetooth", null, AlertType.ERROR);
        al.setTimeout(DemoMIDlet.ALERT_TIMEOUT);
        Display.getDisplay(parent).setCurrent(al, parent.getDisplayable());
    }

    /** Destroys this component. */
    void destroy() {
        // finalize the image server work
        bt_server.destroy();
    }

    /** Gets the image file name from its title (label). */
    String getImageFileName(String imgName) {
        if (imgName == null) {
            return null;
```

```
        }

        // no interface in List to get the index - should find
        int index = -1;

        for (int i = 0; i < imagesList.size(); i++) {
            if (imagesList.getString(i).equals(imgName)) {
                index = i;

                break;
            }
        }

        // not found or not published
        if ((index == -1) || !published[index]) {
            return null;
        }

        return (String)imagesNames.elementAt(index);
    }


    /**
     * Creates the image to indicate the base state.
     */
    private void setupIdicatorImage() {
        // create "on" image
        try {
            onImage = Image.createImage("/images/st-on.png");
        } catch (IOException e) {
            // provide off-screen image then
            onImage = createIndicatorImage(12, 12, 0, 255, 0);
        }

        // create "off" image
        try {
            offImage = Image.createImage("/images/st-off.png");
        } catch (IOException e) {
            // provide off-screen image then
            offImage = createIndicatorImage(12, 12, 255, 0, 0);
        }
    }

    /**
     * Gets the description of images from manifest and
     * prepares the list to control the configuration.
     * <p>
     * The attributes are named "ImageTitle-n" and "ImageImage-n".
     * The value "n" must start at "1" and be incremented by 1.
     */

    private void setupImageList() {
        imagesNames = new Vector();
        imagesList.setCommandListener(this);

        for (int n = 1; n < 100; n++) {
            String name = parent.getAppProperty("ImageName-" + n);

            // no more images available
```

```
            if ((name == null) || (name.length() == 0)) {
                break;
            }

            String label = parent.getAppProperty("ImageTitle-" + n);

            // no label available - use picture name instead
            if ((label == null) || (label.length() == 0)) {
                label = name;
            }

            imagesNames.addElement(name);
            imagesList.append(label, offImage);
        }
    }

    /**
     * Creates the off-screen image with specified size an color.
     */
    private Image createIndicatorImage(int w, int h, int r, int g, int b) {
        Image res = Image.createImage(w, h);
        Graphics gc = res.getGraphics();
        gc.setColor(r, g, b);
        gc.fillRect(0, 0, w, h);

        return res;
    }
} // end of class 'GUIImageServer' definition
```

APPENDIX E: Bluetooth Image Client (J2ME)

```
package example.bluetooth.demo;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Vector;

// jsr082 API
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DataElement;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.UUID;

// midp/cldc API
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.lcdui.Image;


/**
 * Initialize BT device, search for BT services,
 * presents them to user and picks his/her choice,
 * finally download the choosen image and present
 * it to user.
 *
 * @version ,
 */
final class BTImageClient implements Runnable, DiscoveryListener {
    /** Describes this server */
    private static final UUID PICTURES_SERVER_UUID =
        new UUID("F0E0D0C0B0A000908070605040302010", false);

    /** The attribute id of the record item with images names. */
    private static final int IMAGES_NAMES_ATTRIBUTE_ID = 0x4321;

    /** Shows the engine is ready to work. */
    private static final int READY = 0;

    /** Shows the engine is searching bluetooth devices. */
    private static final int DEVICE_SEARCH = 1;

    /** Shows the engine is searching bluetooth services. */
```

```
    private static final int SERVICE_SEARCH = 2;

    /** Keeps the current state of engine. */
    private int state = READY;

    /** Keeps the discovery agent reference. */
    private DiscoveryAgent discoveryAgent;

    /** Keeps the parent reference to process specific actions. */
    private GUIImageClient parent;

    /** Becomes 'true' when this component is finalized. */
    private boolean isClosed;

    /** Process the search/download requests. */
    private Thread processorThread;

    /** Collects the remote devices found during a search. */
    private Vector /* RemoteDevice */ devices = new Vector();

    /** Collects the services found during a search. */
    private Vector /* ServiceRecord */ records = new Vector();

    /** Keeps the device discovery return code. */
    private int discType;

    /** Keeps the services search IDs (just to be able to cancel them). */
    private int[] searchIDs;

    /** Keeps the image name to be load. */
    private String imageNameToLoad;

    /** Keeps the table of {name, Service} to process the user choice. */
    private Hashtable base = new Hashtable();

    /** Informs the thread the download should be canceled. */
    private boolean isDownloadCanceled;

    /** Optimization: keeps service search pattern. */
    private UUID[] uuidSet;

    /** Optimization: keeps attributes list to be retrieved. */
    private int[] attrSet;

    /**
     * Constructs the bluetooth server, but it is initialized
     * in the different thread to "avoid dead lock".
     */
    BTImageClient(GUIImageClient parent) {
        this.parent = parent;

        // we have to initialize a system in different thread...
        processorThread = new Thread(this);
        processorThread.start();
    }

    /**
     * Process the search/download requests.
     */
```

```
    public void run() {
        // initialize bluetooth first
        boolean isBTReady = false;

        try {
            // create/get a local device and discovery agent
            LocalDevice localDevice = LocalDevice.getLocalDevice();
            discoveryAgent = localDevice.getDiscoveryAgent();

            // remember we've reached this point.
            isBTReady = true;
        } catch (Exception e) {
            System.err.println("Can't initialize bluetooth: " + e);
        }

        parent.completeInitialization(isBTReady);

        // nothing to do if no bluetooth available
        if (!isBTReady) {
            return;
        }

        // initialize some optimization variables
        uuidSet = new UUID[2];

        // ok, we are interesting in btspp services only
        uuidSet[0] = new UUID(0x1101);

        // and only known ones, that allows pictures
        uuidSet[1] = PICTURES_SERVER_UUID;

        // we need an only service attribute actually
        attrSet = new int[1];

        // it's "images names" one
        attrSet[0] = IMAGES_NAMES_ATTRIBUTE_ID;

        // start processing the images search/download
        processImagesSearchDownload();
    }

    /**
     * Invoked by system when a new remote device is found -
     * remember the found device.
     */
    public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {
        // same device may found several times during single search
        if (devices.indexOf(btDevice) == -1) {
            devices.addElement(btDevice);
        }
    }

    /**
     * Invoked by system when device discovery is done.
     * <p>
     * Use a trick here - just remember the discType
     * and process its evaluation in another thread.
     */
    public void inquiryCompleted(int discType) {
```

```
        this.discType = discType;

        synchronized (this) {
            notify();
        }
    }

    public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {
        for (int i = 0; i < servRecord.length; i++) {
            records.addElement(servRecord[i]);
        }
    }

    public void serviceSearchCompleted(int transID, int respCode) {
        // first, find the service search transaction index
        int index = -1;

        for (int i = 0; i < searchIDs.length; i++) {
            if (searchIDs[i] == transID) {
                index = i;

                break;
            }
        }

        // error - unexpected transaction index
        if (index == -1) {
            System.err.println("Unexpected transaction index: " + transID);

            // FIXME: process the error case
        } else {
            searchIDs[index] = -1;
        }

        /*
         * Actually, we do not care about the response code -
         * if device is not reachable or no records, etc.
         */

        // make sure it was the last transaction
        for (int i = 0; i < searchIDs.length; i++) {
            if (searchIDs[i] != -1) {
                return;
            }
        }

        // ok, all of the transactions are completed
        synchronized (this) {
            notify();
        }
    }

    /** Sets the request to search the devices/services. */
    void requestSearch() {
        synchronized (this) {
            notify();
        }
    }
```

```
    /** Cancel's the devices/services search. */
    void cancelSearch() {
        synchronized (this) {
            if (state == DEVICE_SEARCH) {
                discoveryAgent.cancelInquiry(this);
            } else if (state == SERVICE_SEARCH) {
                for (int i = 0; i < searchIDs.length; i++) {
                    discoveryAgent.cancelServiceSearch(searchIDs[i]);
                }
            }
        }
    }

    /** Sets the request to load the specified image. */
    void requestLoad(String name) {
        synchronized (this) {
            imageNameToLoad = name;
            notify();
        }
    }

    /** Cancel's the image download. */
    void cancelLoad() {
        /*
         * The image download process is done by
         * this class's thread (not by a system one),
         * so no need to wake up the current thread -
         * it's running already.
         */
        isDownloadCanceled = true;
    }

    /**
     * Destroy a work with bluetooth - exits the accepting
     * thread and close notifier.
     */
    void destroy() {
        synchronized (this) {
            isClosed = true;
            isDownloadCanceled = true;
            notify();

            // FIXME: implement me
        }

        // wait for acceptor thread is done
        try {
            processorThread.join();
        } catch (InterruptedException e) {
        } // ignore
    }

    /**
     * Processes images search/download until component is closed
     * or system error has happen.
     */
    private synchronized void processImagesSearchDownload() {
        while (!isClosed) {
            // wait for new search request from user
```

```
            state = READY;

            try {
                wait();
            } catch (InterruptedException e) {
                System.err.println("Unexpected interruption: " + e);

                return;
            }

            // check the component is destroyed
            if (isClosed) {
                return;
            }

            // search for devices
            if (!searchDevices()) {
                return;
            } else if (devices.size() == 0) {
                continue;
            }

            // search for services now
            if (!searchServices()) {
                return;
            } else if (records.size() == 0) {
                continue;
            }

            // ok, something was found - present the result to user now
            if (!presentUserSearchResults()) {
                // services are found, but no names there
                continue;
            }

            // the several download requests may be processed
            while (true) {
                // this download is not canceled, right?
                isDownloadCanceled = false;

                // ok, wait for download or need to wait for next search
                try {
                    wait();
                } catch (InterruptedException e) {
                    System.err.println("Unexpected interruption: " + e);

                    return;
                }

                // check the component is destroyed
                if (isClosed) {
                    return;
                }

                // this means "go to the beginning"
                if (imageNameToLoad == null) {
                    break;
                }
```

```
                // load selected image data
                Image img = loadImage();

                // FIXME: this never happen - monitor is taken...
                if (isClosed) {
                    return;
                }

                if (isDownloadCanceled) {
                    continue; // may be next image to be download
                }

                if (img == null) {
                    parent.informLoadError("Can't load image: " + imageNameToLoad);

                    continue; // may be next image to be download
                }

                // ok, show image to user
                parent.showImage(img, imageNameToLoad);

                // may be next image to be download
                continue;
            }
        }
    }

    /**
     * Search for bluetooth devices.
     *
     * @return false if should end the component work.
     */
    private boolean searchDevices() {
        // ok, start a new search then
        state = DEVICE_SEARCH;
        devices.removeAllElements();

        try {
            discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
        } catch (BluetoothStateException e) {
            System.err.println("Can't start inquiry now: " + e);
            parent.informSearchError("Can't start device search");

            return true;
        }

        try {
            wait(); // until devices are found
        } catch (InterruptedException e) {
            System.err.println("Unexpected interruption: " + e);

            return false;
        }

        // this "wake up" may be caused by 'destroy' call
        if (isClosed) {
            return false;
        }
```

```
        // no?, ok, let's check the return code then
        switch (discType) {
        case INQUIRY_ERROR:
            parent.informSearchError("Device discovering error...");

        // fall through
        case INQUIRY_TERMINATED:
            // make sure no garbage in found devices list
            devices.removeAllElements();

            // nothing to report - go to next request
            break;

        case INQUIRY_COMPLETED:

            if (devices.size() == 0) {
                parent.informSearchError("No devices in range");
            }

            // go to service search now
            break;

        default:
            // what kind of system you are?... :(
            System.err.println("system error:" + " unexpected device discovery code: " +
discType);
            destroy();

            return false;
        }

        return true;
    }

    /**
     * Search for proper service.
     *
     * @return false if should end the component work.
     */
    private boolean searchServices() {
        state = SERVICE_SEARCH;
        records.removeAllElements();
        searchIDs = new int[devices.size()];

        boolean isSearchStarted = false;

        for (int i = 0; i < devices.size(); i++) {
            RemoteDevice rd = (RemoteDevice)devices.elementAt(i);

            try {
                searchIDs[i] = discoveryAgent.searchServices(attrSet, uuidSet, rd, this);
            } catch (BluetoothStateException e) {
                System.err.println("Can't search services for: " + rd.getBluetoothAddress()
+
                    " due to " + e);
                searchIDs[i] = -1;

                continue;
            }
```

```
            isSearchStarted = true;
        }

        // at least one of the services search should be found
        if (!isSearchStarted) {
            parent.informSearchError("Can't search services.");

            return true;
        }

        try {
            wait(); // until services are found
        } catch (InterruptedException e) {
            System.err.println("Unexpected interruption: " + e);

            return false;
        }

        // this "wake up" may be caused by 'destroy' call
        if (isClosed) {
            return false;
        }

        // actually, no services were found
        if (records.size() == 0) {
            parent.informSearchError("No proper services were found");
        }

        return true;
    }

    /**
     * Gets the collection of the images titles (names)
     * from the services, prepares a hashtable to match
     * the image name to a services list, presents the images names
     * to user finally.
     *
     * @return false if no names in found services.
     */
    private boolean presentUserSearchResults() {
        base.clear();

        for (int i = 0; i < records.size(); i++) {
            ServiceRecord sr = (ServiceRecord)records.elementAt(i);

            // get the attribute with images names
            DataElement de = sr.getAttributeValue(IMAGES_NAMES_ATTRIBUTE_ID);

            if (de == null) {
                System.err.println("Unexpected service - missed attribute");

                continue;
            }

            // get the images names from this attribute
            Enumeration deEnum = (Enumeration)de.getValue();

            while (deEnum.hasMoreElements()) {
```

```
                de = (DataElement)deEnum.nextElement();

                String name = (String)de.getValue();

                // name may be stored already
                Object obj = base.get(name);

                // that's either the ServiceRecord or Vector
                if (obj != null) {
                    Vector v;

                    if (obj instanceof ServiceRecord) {
                        v = new Vector();
                        v.addElement(obj);
                    } else {
                        v = (Vector)obj;
                    }

                    v.addElement(sr);
                    obj = v;
                } else {
                    obj = sr;
                }

                base.put(name, obj);
            }
        }

        return parent.showImagesNames(base);
    }

    /**
     * Loads selected image data.
     */
    private Image loadImage() {
        if (imageNameToLoad == null) {
            System.err.println("Error: imageNameToLoad=null");

            return null;
        }

        // ok, get the list of service records
        ServiceRecord[] sr = null;
        Object obj = base.get(imageNameToLoad);

        if (obj == null) {
            System.err.println("Error: no record for: " + imageNameToLoad);

            return null;
        } else if (obj instanceof ServiceRecord) {
            sr = new ServiceRecord[] { (ServiceRecord)obj };
        } else {
            Vector v = (Vector)obj;
            sr = new ServiceRecord[v.size()];

            for (int i = 0; i < v.size(); i++) {
                sr[i] = (ServiceRecord)v.elementAt(i);
            }
        }
```

```
        // now try to load the image from each services one by one
        for (int i = 0; i < sr.length; i++) {
            StreamConnection conn = null;
            String url = null;

            // the process may be canceled
            if (isDownloadCanceled) {
                return null;
            }

            // first - connect
            try {
                url = sr[i].getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT,
false);
                conn = (StreamConnection)Connector.open(url);
            } catch (IOException e) {
                System.err.println("Note: can't connect to: " + url);

                // ignore
                continue;
            }

            // then open a steam and write a name
            try {
                OutputStream out = conn.openOutputStream();
                out.write(imageNameToLoad.length()); // length is 1 byte
                out.write(imageNameToLoad.getBytes());
                out.flush();
                out.close();
            } catch (IOException e) {
                System.err.println("Can't write to server for: " + url);

                // close stream connection
                try {
                    conn.close();
                } catch (IOException ee) {
                } // ignore

                continue;
            }

            // then open a steam and read an image
            byte[] imgData = null;

            try {
                InputStream in = conn.openInputStream();

                // read a length first
                int length = in.read() << 8;
                length |= in.read();

                if (length <= 0) {
                    throw new IOException("Can't read a length");
                }

                // read the image now
                imgData = new byte[length];
                length = 0;
```

```
                while (length != imgData.length) {
                    int n = in.read(imgData, length, imgData.length - length);

                    if (n == -1) {
                        throw new IOException("Can't read a image data");
                    }

                    length += n;
                }

                in.close();
            } catch (IOException e) {
                // SEMC BEGIN:
                System.err.println("Can't read from server for: " + url + ". Message was: "
+ e);
                // SEMC END
                continue;
            } finally {
                // close stream connection anyway
                try {
                    conn.close();
                } catch (IOException e) {
                } // ignore
            }

            // ok, may it's a chance
            Image img = null;

            try {
                img = Image.createImage(imgData, 0, imgData.length);
            } catch (Exception e) {
                // may be next time
                System.err.println("Error: wrong image data from: " + url);

                continue;
            }

            return img;
        }

        return null;
    }
} // end of class 'BTImageClient' definition
```

APPENDIX F: GUI Image Client (J2ME)

```
package example.bluetooth.demo;

import java.io.IOException;
import java.util.Enumeration;
import java.util.Hashtable;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Gauge;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.ImageItem;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.StringItem;


/**
 * Provides a GUI to present the download options
 * to used, gives a chance to make a choice,
 * finally shows the downloaded image.
*/
final class GUIImageClient implements CommandListener {
    /** This command goes to demo main screen. */
    private final Command SCR_MAIN_BACK_CMD = new Command("Back", Command.BACK, 2);

    /** Starts the proper services search. */
    private final Command SCR_MAIN_SEARCH_CMD = new Command("Find", Command.OK, 1);

    /** Cancels the device/services discovering. */
    private final Command SCR_SEARCH_CANCEL_CMD = new Command("Cancel", Command.BACK, 2);

    /** This command goes to client main screen. */
    private final Command SCR_IMAGES_BACK_CMD = new Command("Back", Command.BACK, 2);

    /** Start the chosen image download. */
    private final Command SCR_IMAGES_LOAD_CMD = new Command("Load", Command.OK, 1);

    /** Cancels the image download. */
    private final Command SCR_LOAD_CANCEL_CMD = new Command("Cancel", Command.BACK, 2);

    /** This command goes from image screen to images list one. */
    private final Command SCR_SHOW_BACK_CMD = new Command("Back", Command.BACK, 2);

    /** The main screen of the client part. */
    private final Form mainScreen = new Form("Image Viewer");

    /** The screen with found images names. */
```

```
    private final List listScreen = new List("Image Viewer", List.IMPLICIT);

    /** The screen with download image. */
    private final Form imageScreen = new Form("Image Viewer");

    /** Keeps the parent MIDlet reference to process specific actions. */
    private DemoMIDlet parent;

    /** This object handles the real transmission. */
    private BTImageClient bt_client;

    /** Constructs client GUI. */
    GUIImageClient(DemoMIDlet parent) {
        this.parent = parent;
        bt_client = new BTImageClient(this);
        mainScreen.addCommand(SCR_MAIN_BACK_CMD);
        mainScreen.addCommand(SCR_MAIN_SEARCH_CMD);
        mainScreen.setCommandListener(this);
        listScreen.addCommand(SCR_IMAGES_BACK_CMD);
        listScreen.addCommand(SCR_IMAGES_LOAD_CMD);
        listScreen.setCommandListener(this);
        imageScreen.addCommand(SCR_SHOW_BACK_CMD);
        imageScreen.setCommandListener(this);
    }

    /**
     * Process the command events.
     *
     * @param c - the issued command.
     * @param d - the screen object the command was issued for.
     */
    public void commandAction(Command c, Displayable d) {
        // back to demo main screen
        if (c == SCR_MAIN_BACK_CMD) {
            destroy();
            parent.show();

            return;
        }

        // starts images (device/services) search
        if (c == SCR_MAIN_SEARCH_CMD) {
            Form f = new Form("Searching...");
            f.addCommand(SCR_SEARCH_CANCEL_CMD);
            f.setCommandListener(this);
            f.append(new Gauge("Searching images...", false, Gauge.INDEFINITE,
                    Gauge.CONTINUOUS_RUNNING));
            Display.getDisplay(parent).setCurrent(f);
            bt_client.requestSearch();

            return;
        }

        // cancels device/services search
        if (c == SCR_SEARCH_CANCEL_CMD) {
            bt_client.cancelSearch();
            Display.getDisplay(parent).setCurrent(mainScreen);

            return;
```

```
        }

        // back to client main screen
        if (c == SCR_IMAGES_BACK_CMD) {
            bt_client.requestLoad(null);
            Display.getDisplay(parent).setCurrent(mainScreen);

            return;
        }

        // starts image download
        if (c == SCR_IMAGES_LOAD_CMD) {
            Form f = new Form("Loading...");
            f.addCommand(SCR_LOAD_CANCEL_CMD);
            f.setCommandListener(this);
            f.append(new Gauge("Loading image...", false, Gauge.INDEFINITE,
Gauge.CONTINUOUS_RUNNING));
            Display.getDisplay(parent).setCurrent(f);

            List l = (List)d;
            bt_client.requestLoad(l.getString(l.getSelectedIndex()));

            return;
        }

        // cancels image load
        if (c == SCR_LOAD_CANCEL_CMD) {
            bt_client.cancelLoad();
            Display.getDisplay(parent).setCurrent(listScreen);

            return;
        }

        // back to client main screen
        if (c == SCR_SHOW_BACK_CMD) {
            Display.getDisplay(parent).setCurrent(listScreen);

            return;
        }
    }

    /**
     * We have to provide this method due to "do not do network
     * operation in command listener method" restriction, which
     * is caused by crooked midp design.
     *
     * This method is called by BTImageClient after it is done
     * with bluetooth initialization and next screen is ready
     * to appear.
     */
    void completeInitialization(boolean isBTReady) {
        // bluetooth was initialized successfully.
        if (isBTReady) {
            StringItem si = new StringItem("Ready for images search!",
null);
            si.setLayout(StringItem.LAYOUT_CENTER |
StringItem.LAYOUT_VCENTER);
            mainScreen.append(si);
            Display.getDisplay(parent).setCurrent(mainScreen);

            return;
        }

        // something wrong
        Alert al = new Alert("Error", "Can't initialize bluetooth", null,
AlertType.ERROR);
        al.setTimeout(DemoMIDlet.ALERT_TIMEOUT);
        Display.getDisplay(parent).setCurrent(al, parent.getDisplayable());
```

```
    }

    /** Destroys this component. */
    void destroy() {
        // finalize the image client work
        bt_client.destroy();
    }

    /**
     * Informs the error during the images search.
     */
    void informSearchError(String resMsg) {
        Alert al = new Alert("Error", resMsg, null, AlertType.ERROR);
        al.setTimeout(DemoMIDlet.ALERT_TIMEOUT);
        Display.getDisplay(parent).setCurrent(al, mainScreen);
    }

    /**
     * Informs the error during the selected image load.
     */
    void informLoadError(String resMsg) {
        Alert al = new Alert("Error", resMsg, null, AlertType.ERROR);
        al.setTimeout(DemoMIDlet.ALERT_TIMEOUT);
        Display.getDisplay(parent).setCurrent(al, listScreen);
    }

    /**
     * Shows the downloaded image.
     */
    void showImage(Image img, String imgName) {
        imageScreen.deleteAll();
        imageScreen.append(new ImageItem(imgName, img,
                ImageItem.LAYOUT_CENTER | ImageItem.LAYOUT_VCENTER,
"Downloaded image: " + imgName));
        Display.getDisplay(parent).setCurrent(imageScreen);
    }

    /**
     * Shows the available images names.
     * @return false if no images names were found actually
     */
    boolean showImagesNames(Hashtable base) {
        Enumeration keys = base.keys();

        // no images actually
        if (!keys.hasMoreElements()) {
            informSearchError("No images names in found services");

            return false;
        }

        // prepare the list to be shown
        while (listScreen.size() != 0) {
            listScreen.delete(0);
        }

        while (keys.hasMoreElements()) {
            listScreen.append((String)keys.nextElement(), null);
        }

        Display.getDisplay(parent).setCurrent(listScreen);

        return true;
    }
} // end of class 'GUIImageClient' definition
```