GUI ABSTRACTION OF A SENSOR FIELD ON MOBILE DEVICE

by

GAURAV CHAUHAN

B.E., R.G.P.V. BHOPAL 2006

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2010

Approved by:

Major Professor
Dr. Gurdip Singh

Abstract

A sensor network can be used to observe events performed by physical entities and their physical locations. The growing need of wireless sensor networks to monitor different events can be accomplished by tiny computing platforms called Motes. Having a GUI abstraction of the region can help in getting a heads up display of the region which can be used for several purposes, e.g. in hospitals for tracking different events in case of a fire emergency. It can be helpful for firefighters entering a large building by providing them prior information of a building layout as it is difficult to see through due to heavy smoke. This project develops an approach to show via GUI information of a region with the help of motes transferring data wirelessly. It is a 3-tier application including a server, a mobile client and a mote setup. The motes execute TinyOS, which is specifically designed operating system for sensor networks.

The project has been tested in the computer science department building, Crossbow's TelosB motes have been used for the mote setup. The programs for the motes are written in nesC (dialect of C), and the client and the server program are written in Java.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I owe my deepest gratitude to my advisor, Dr. Gurdip Singh, whose encouragement, guidance, and support from the initial to the final level enabled me to develop an understanding of the subject. Without his guidance and persistent help it would have been difficult to complete this report.

I am thankful to Dr. Daniel Andresen and Dr. Mitchell Neilsen for kindly serving in my major committee.

Special thanks to Sumeet Gujrati for his useful advice and suggestions and to all of my friends who have supported me throughout the project.

Finally I wish to thank my parents, Vikram Singh Chauhan and Savita Chauhan who have always supported and encouraged me to do my best in all matters of life. Without them this would have never been possible.

CHAPTER 1 - Introduction

Wireless sensor networks are being used widely for various applications. They are networks of autonomous distributed sensors which can perform various useful tasks like sensing temperature, light intensity, sound, vibration, motion, pressure and many other physical quantities. The development of sensor networks was motivated by military applications such as surveillance of territory. But now sensor networks are being used in several applications such as day to day industrial applications, health care, environment, and inhabitant monitoring. Location enhanced application is one of the most enhanced applications used today. It can help in detecting positions of people, places, and things. Despite so much use of the location tracking technology and steady improvement over the years, they are still complex to use. For example, if we use GPS for location tracking then it will not work indoors or in an area with high density of buildings because of signal problem.

In RFID approach, RSSI (Received Signal Strength Indicator) is used for tracking the RSSI signals which are too sensitive to the direction of RFID reader’s antenna, the orientation of RFID tag, human interface, and distraction in the propagations media present. Mote track [3] is an application which works using RFID signals and helps in location tracking with RSSI signals. However, the method is not 100% accurate and also requires at least 10 to 12 motes. Setting up of mote track application requires two steps for location tracking: The first step involves collecting radio signals around the territory and the next step locates the position. It might happen that in the data collection approach, some of the packets are lost. In that case, there will be inaccuracy in location tracking.

Our project is an approach for displaying the position of the devices with the help of motes in a given building. The software is a 3 tier application; the setup includes motes, a server and a mobile client. Beacon motes are setup in different part of a room or are attached with devices present in the room and they send their coordinates to the base station. The server contains the images of various rooms in the building and can be placed anywhere. A mobile client has a mote attached to it which acts as a base station and listens to the beacon motes in the vicinity. The base station sends the message to the mobile client and then the mobile client displays the image of the room or territory with the positions of the motes and the ID of the devices which are placed near the Beacon motes.

1.1 Motes

Wireless sensor networks use motes for performing various tasks like sensing and location tracking. Motes are an integrated part of wireless sensor network. They also known as sensor nodes. Motes perform various tasks such as processing, gathering information and communicating with other motes in the network. It has various components such as microcontroller, external memory, transceiver, power source, and sensors. Various kinds of motes are available. In this project, we have used the Telosb mote which has a USB interface for programming and collecting data. It has an integrated onboard antenna.

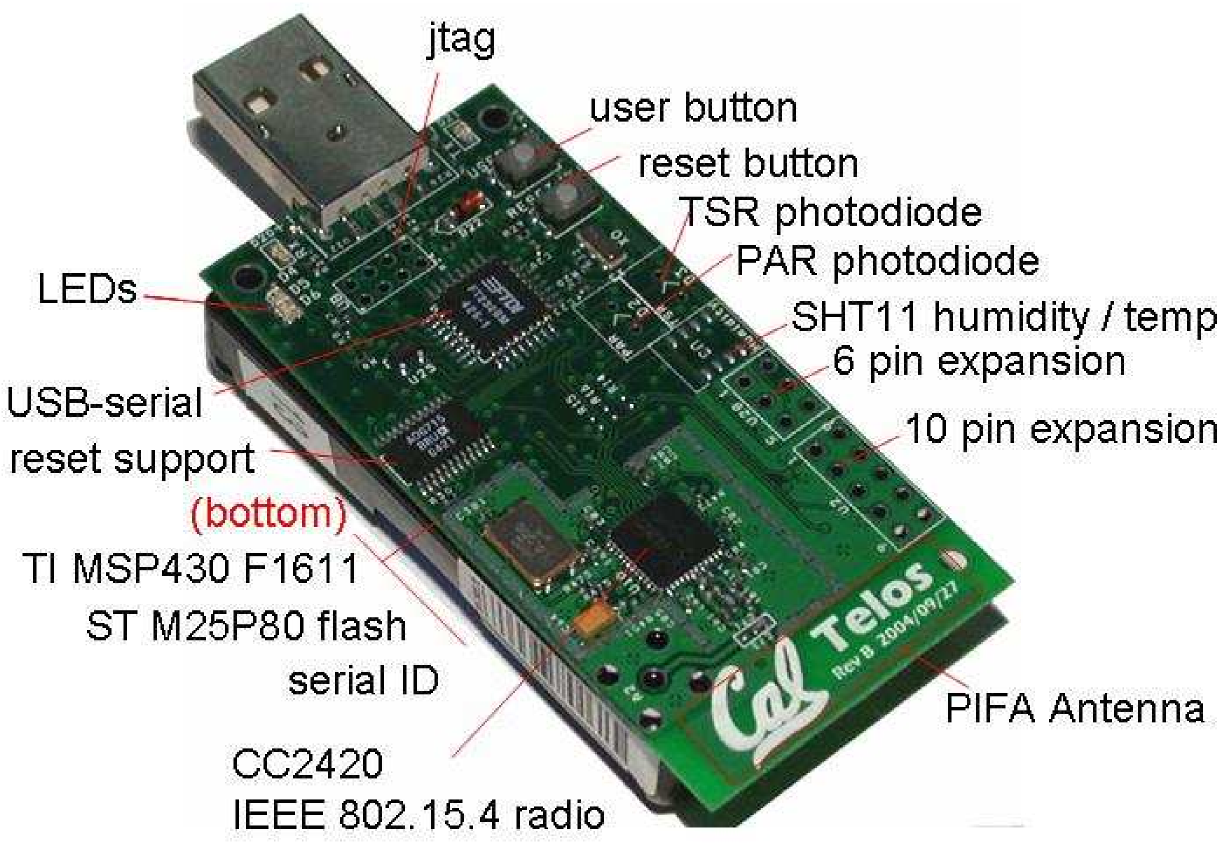


Figure 1.1 Telosb Mote

1.2 Server

Server can be any device which has JVM and Internet capability in it. In this project laptop is used as the server. Server stores maps of different rooms in image format(.jpg or .jpeg). It sends the room maps to client according to the room identification received from client.

1.3 Mobile Client

Mobile client should have JVM and Internet capability in it. Client should have a USB port so that we can connect base station (mote) with it. Mobile client receive the room map from server and displays the coordinates of motes present in room on room map.

This report is organized as follows –

1. Chapter 2 explains the message structure of Tinyos and also the communication layout of Tinyos. Section 2.2 contains the code structure for sending and receiving the message in the project.
2. Chapter 3 explains the design consideration for building the project.
3. Chapter 4 explains the implementation of the project. Section 4.1 explains the mote deployment in the project. Section 4.2 explains how to implement server and client it also have detailed explanation for running IPAQ Java and Telosb.
4. Chapter 5 explains the performance and testing of the project with the results of test performance.

CHAPTER 2 - Message communication in sensor network

Message communication in a sensor network is done via radio communication. It uses the Active Message model (AM). In this approach, each packet in the network specifies a handler ID that will be involve the recipient nodes. When the receiver node receives the message, it will invoke the event associated with that handler ID. The default message structure is defined in the directory of Tinyos under ….tos/type/AM.h and appears as shown below:

2.1 Message Structure

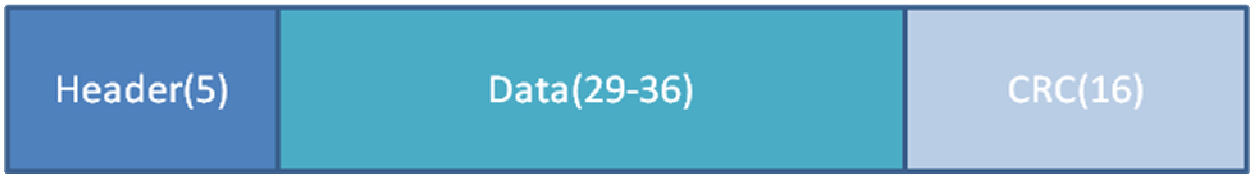


Figure 2.1 Message Structure

Header

2-byte Destination address

1-byte Type (AM handler ID)

1-byte Group

1-byte Length

Data

CRC Checksum

Each of these fields is explained below:

- Destination Address

  It identifies an individual mote similar to the way MAC Address does for the computer systems.

  It is 2 bytes long.

  Reserved Addresses: 0x007E –UART (TOS_UART_ADDR)

  0xFFFF –Radio broadcast (TOS_BCAST_ADDR)

  Local address

  Addressed with local constant TOS_LOCAL_ADDRESS

- Type

  On receipt of a message, the receiving mote activates an event that is associated with the handler ID

  Different motes can associate DIFFERENT receive events with the same handler ID

  1 byte

  Active Message Handler ID

- Group

  It allows multiple “groups” of motes to share same channel just the way a Work Group operates in an actual network.

  Default is 0x7D

  Set by defining preprocessor symbol (in …/apps/Make local)

  DEFAULT_LOCAL_GROUP

2.2 Message communication between motes.

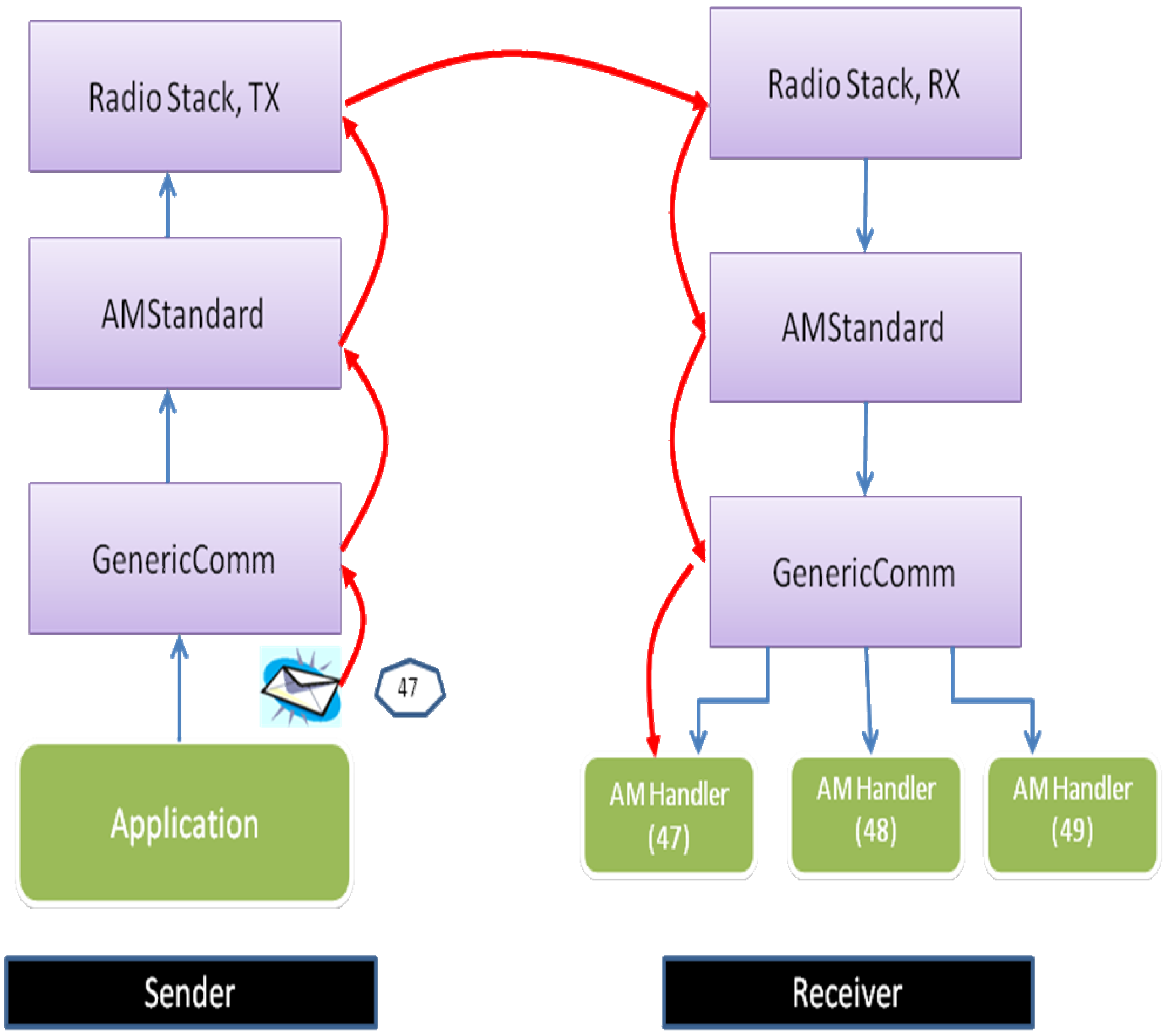


Figure 2.2 Message communication layout

2.2.1 Sending a message:

2.2.1.1 Content of configuration file:

```
configuration BeaconMote
{
}
implementation
{
   components Main, BeaconMoteM, GenericComm as Comm, LedsC as Leds, TimerC,
#if defined(PLATFORM_MICA2) || defined(PLATFORM_MICA2DOT)
   CC1000ControlM;
   BeaconMoteM.CC1000Control -> CC1000ControlM.CC1000Control;
#else // assume CC2420 radio (e.g. MicaZ, Telos)
   CC2420ControlM;
   BeaconMoteM.CC2420Control -> CC2420ControlM.CC2420Control;
#endif
   Main.StdControl -> Comm.Control;
   Main.StdControl -> BeaconMoteM;
   Main.StdControl -> TimerC;
   BeaconMoteM.Timer -> TimerC.Timer[unique("Timer")];
   BeaconMoteM.SendMsg -> Comm.SendMsg[AM_BEACONMSG];
   BeaconMoteM.Leds -> Leds;
}
```

2.2.1.2 Algorithm of Module file:

```
boolpending;
structTOS_Msgdata;
command result_tIntOutput.output(uint16_t value) {
IntMsg*message = (IntMsg*)data.data;
if (!pending) {
pending = TRUE;
message->val= value;
atomic {
message->src= TOS_LOCAL_ADDRESS;
}
if (call Send.send(TOS_BCAST_ADDR, sizeof(IntMsg), &data))
return SUCCESS;
pending = FALSE;
}
return FAIL;
}
```

2.2.2 Receiving a Message

For receiving any message the mote performs the following steps:

- Message arrives and is placed in a buffer
- Active Message layer decodes the handler type and dispatches the message
- The buffer is handed to the application component through the receive() event of the ReceiveMsg interface (tos/interfaces/ReceiveMsg.nc)
- The application component returns a pointer to a buffer upon completion

2.2.2.1 Content of configuration file:

```
configuration MobileMote
{
}
implementation
{
   components Main, MobileMoteM, TimerC, LedsC, GenericComm as Comm,
#if defined(PLATFORM_MICA2) || defined(PLATFORM_MICA2DOT)
   CC1000ControlM;
```

```
    MobileMoteM.CC1000Control -> CC1000ControlM.CC1000Control;
#else // assume CC2420 radio (e.g. MicaZ, Telos)
    CC2420ControlM;
    MobileMoteM.CC2420Control -> CC2420ControlM.CC2420Control;
#endif


    Main.StdControl -> MobileMoteM.StdControl;
    Main.StdControl -> Comm.Control;
    Main.StdControl -> TimerC;
    MobileMoteM.Leds -> LedsC.Leds;
    MobileMoteM.Timer_ChangeFreqChan -> TimerC.Timer[unique("Timer")];
    MobileMoteM.Timer_EstLoc -> TimerC.Timer[unique("Timer")];
   MobileMoteM.ReceiveMsg -> Comm.ReceiveMsg[AM_BEACONMSG];
}
```

2.2.2.2 Content of Module file:

```
event TOS_MsgPtrReceiveIntMsg.receive(TOS_MsgPtrm) {
IntMsg*message = (IntMsg*)m->data;
call IntOutput.output(message->val);
return m;
}
```

CHAPTER 3 - Design Consideration

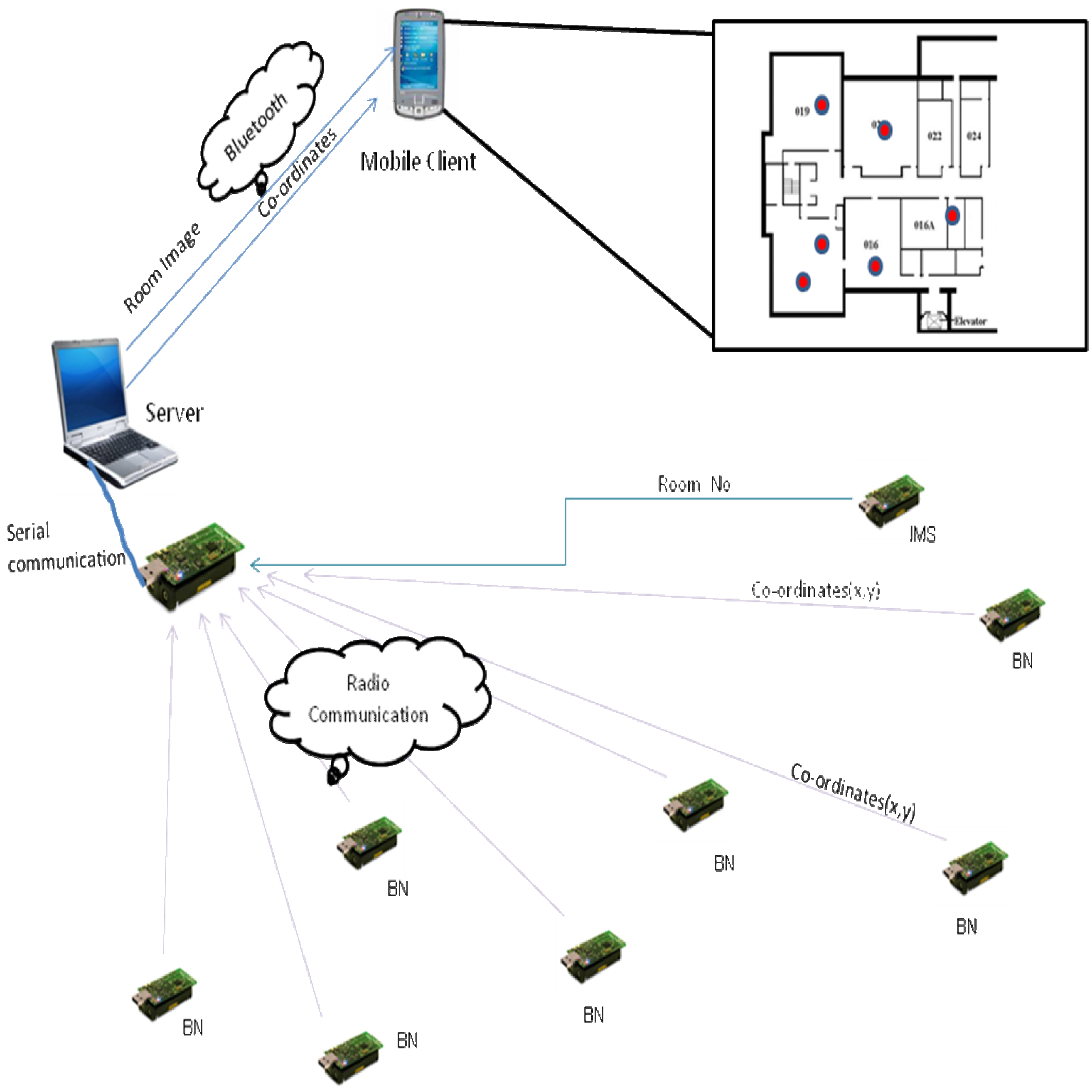
This chapter describes the three approaches which were implemented for solving the problem. For designing the GUI of a room with the coordinates of various devices, we divided the project in three parts: mote side setup, server side, and the client side. In these three parts, the message sending techniques are different as also the method chosen for the communication between client, server and motes. Out of the three approaches, the third approach seems to be the most efficient.

3.1 Initial approach (Bluetooth)

The very first approach used the Bluetooth technology for transferring data from the server to the client and vice versa. The motes are placed in the room or attached to various devices. The Beacon motes send their coordinates to the base station which is attached to the server. There is one mote placed in the room which has the responsibility of identifying the room. This mote sends a variable called Image Token to the base station attached to the server. The server sends that variable value with the help of serial port to the base station. According to the Image Token variable value received, the server then sends the image of this room to the mobile client via Bluetooth. Once the image is transferred, the base station starts accepting the coordinates from the various Beacon motes and it transfers these coordinates to the client through the server. On the client side, the coordinates are received and displayed on the image received previously from the server.

3.1.1 Need for change

The server and the client contact each other via Bluetooth. The server initiates the connection with the client for passing the image and other messages received from the motes. The Bluetooth connection is not very consistent since it often fails to get established with the client when it is required to send data and there is a high rate of message loss. Bluetooth has a constraint of the short range for message transfer. To avoid this drawback, TCP/IP connection approach is used.



IMS – Image token Sender
BN – Beacon Node

Figure 3.1 First approach (Bluetooth)

3.2 Second approach

The project is divided in 3 different parts, the first part contains the setup of motes and the coding is done in nesC (dialect in C). The second part is the server design with the coding being done in Java. The third part is the client and it also has its coding done in Java.

3.2.1 Mote code (nesC code)

The mote deployment consists of three different kinds of code: Beacon motes code, Image Token code, and base station code. We need to change the message format of Tinyos message structure for all the three codes.

3.2.1.1 Beacon motes

The Beacon motes contain coordinates of the location where they are placed. These motes send signals repeatedly at regular time intervals to avoid any message loss. If the mote is attached to a device, the device ID should be sent. The messages are being sent to a base station which is attached to the server. For sending the coordinates, we need to change the message structure in Tinyos. The original message structure (TOS message structure) looks like this:

```
typedef struct TOS_Msg
{
  /* The following fields are transmitted/received on the radio. */
  uint16_t addr;
  uint8_t type;
  uint8_t group;
  uint8_t length;
  int8_t data[TOSH_DATA_LENGTH];
  uint16_t crc;

  /* The following fields are not actually transmitted or received
   * on the radio! They are used for internal accounting only.
   * The reason they are in this structure is that the AM interface
   * requires them to be part of the TOS_Msg that is passed to
   * send/receive operations.
   */
  uint16_t strength;
  uint8_t ack;
  uint16_t time;
  uint8_t sendSecurityMode;
  uint8_t receiveSecurityMode;
} TOS_Msg;
```

The message structure for Beacon motes:

```
struct EventMsgsend
{
int src_Addr;
int group_id;
int device_id;
char x_coordinate[2];
char y_coordinate[2];
};
typedef struct EventMsgsend EventMsgsend;
typedef EventMsgsend* EventMsgsendPtr;
```

3.2.1.2 Image Token mote

This mote is responsible for sending the identifier of the room to the base station. This room number is eventually received by the server for sending the image of room map to the mobile client for localization. The message structure used for sending the room ID is:

```
struct EventMsgsend
{
int src_Addr;
int group_id;
int device_id;
char roomnumber[2];
};
typedef struct EventMsgsend EventMsgsend;
typedef EventMsgsend* EventMsgsendPtr;
```

3.2.1.3 Base Station

The base station is attached to the server and receives the signal from the Image token mote and the Beacon motes. The base station first receives the signal from the Image token mote which triggers the receiver function for receiving messages from the Beacon motes. The base station is attached with the server through a USB port; it sends all the signals to the server through the serial port.

3.2.2 Server side code

The coding on the server is done in Java. The server receives the message signal from the base station attached to it. The signal communication between motes is in hexadecimal format and the sender adds its own header in the start of every message. We need to take care of the baud rate and for the telosb, it should be 56700. The server code has a parser to convert the signals coming from the mote in a readable format and truncates the unnecessary data from the mote. The server reads the message coming from the Image Token mote and sends the image to the mobile client with the help of TCP/IP connection. After image transfer, the server starts sending the coordinates received from the Beacon motes to the mobile client.

3.2.3 Client side code

The coding on the client side is done in Java. The mobile client receives the data from the server and displays the location on the image received from the server. The client code makes sure that it provides the mobility and continues to receive messages from the server when it moves from place to place.

3.2.4 Need for change

The mentioned approach has the disadvantage of high message traffic. All the messages from the Beacon motes and the Image Token mote are first transferred to the base station and then to the server and from the server to the client. It also puts the constraint for the server to be present in the room. To avoid the above problems, another design approach is followed which is much more efficient than this approach.

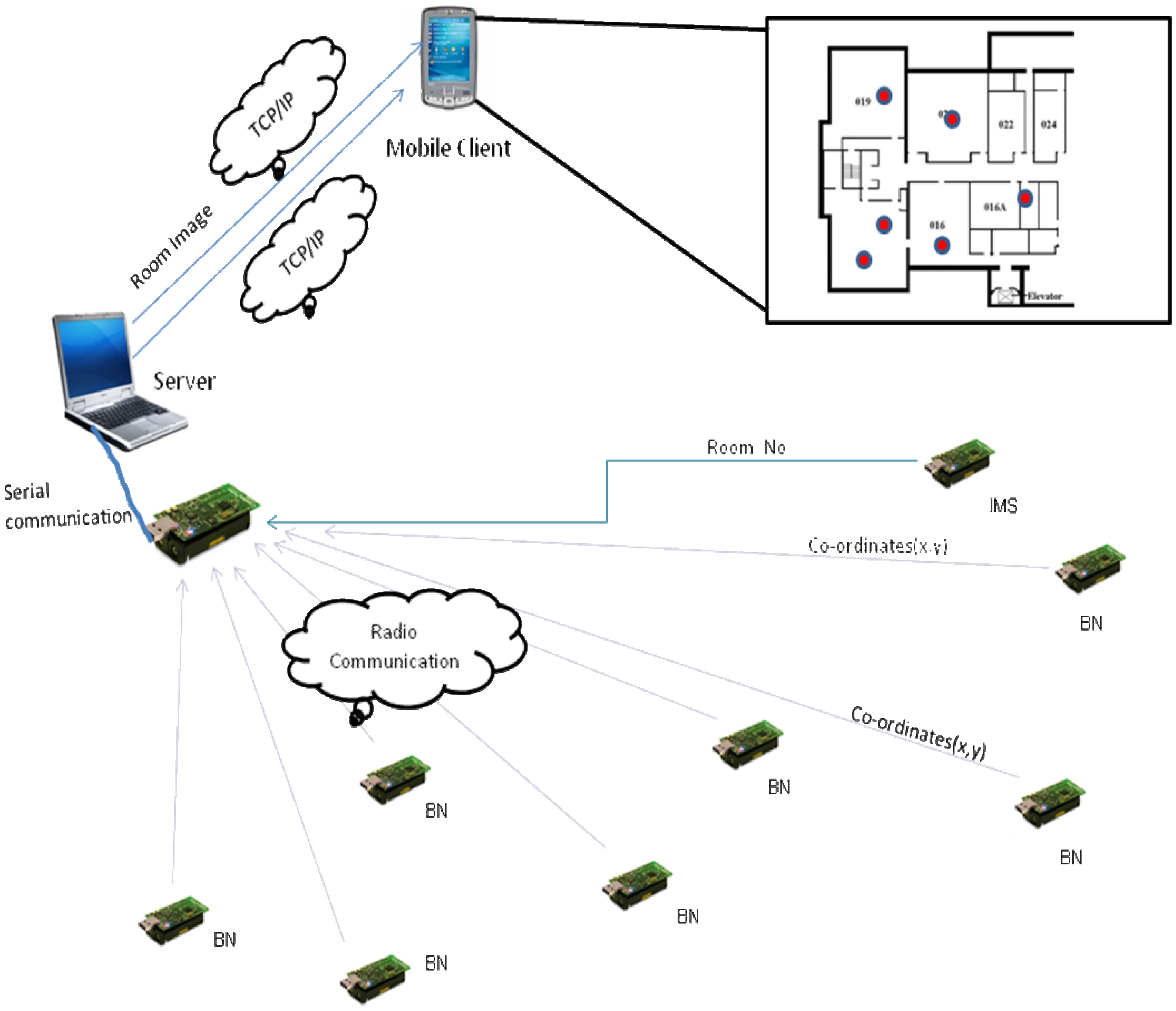


Figure 3.2 Second approach (TCP/IP)

3.3 Third Approach

The design structure in this approach is better than the previous approaches and it decreases the traffic congestion.

3.3.1 Motecode

The mote code remains the same as before. The Image Token mote, the Beacon mote and the base station are still deployed with the same code.

3.3.2 Mobile client

In this design approach, we attach the base station with the mobile client and this client receives all the messages from the Image Token mote and the Beacon motes. When the mobile client comes in the vicinity of the Image Token mote, it receives the room number from the Image Token mote. It then establishes the socket connection with the server and sends the image token variable value to the server. The mobile client waits for a response from the server. As soon as it gets the image, it starts listening on the serial port for coordinates from the Beacon motes through the base station attached to it. The parser changes the signal from the motes in a readable format and displays the coordinates on the image of room.

```
08 00 00 00 00 00 7e 00 84 7d 17 22 01 f4 03 03 00 00
08 00 00 00 00 00 7e 00 84 7d 14 2c 01 00 e7 03 00 00
08 00 00 00 00 00 7e 00 84 7d 07 bc 02 00 03 e7 00 00
08 00 00 00 00 00 7e 00 84 7d 0c 2b 02 00 e7 03 00 00
08 00 00 00 00 00 7e 00 84 7d 0a c7 00 00 00 ee 00 00
```

Figure 3.3 Received message from Mote

The message from the motes is then changed into the decimal format and it looks as follows

```
Device ID: 23--->  X coordinate: 298 ---- Y coordinates: 1012
Device ID: 20--->  X coordinate: 300 ---- Y coordinates: 356
Device ID: 07--->  X coordinate: 700 ---- Y coordinates: 70
Device ID: 12--->  X coordinate: 555 ---- Y coordinates: 626
Device ID: 10--->  X coordinate: 199 ---- Y coordinates: 435
```

Figure 3.4 Changed message format after parsing

3.3.3 Server Side

The server receives the room number from the mobile client through socket connection and sends the image of the room to the mobile client, instead of getting it from the Image Token mote. This design approach removes the requirement of the server to be present in the vicinity of the motes.

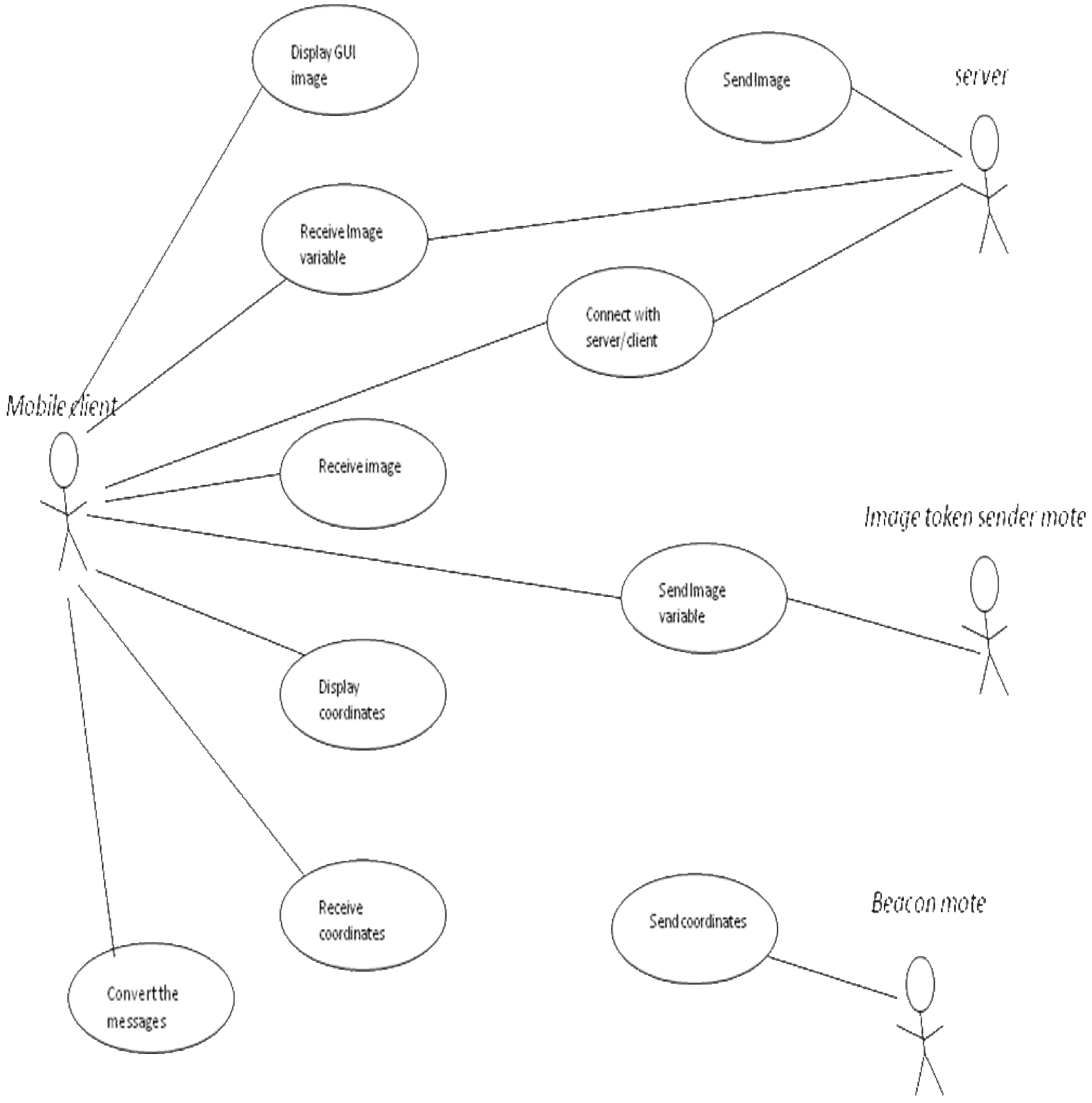


Figure 3.5 Use case Diagram

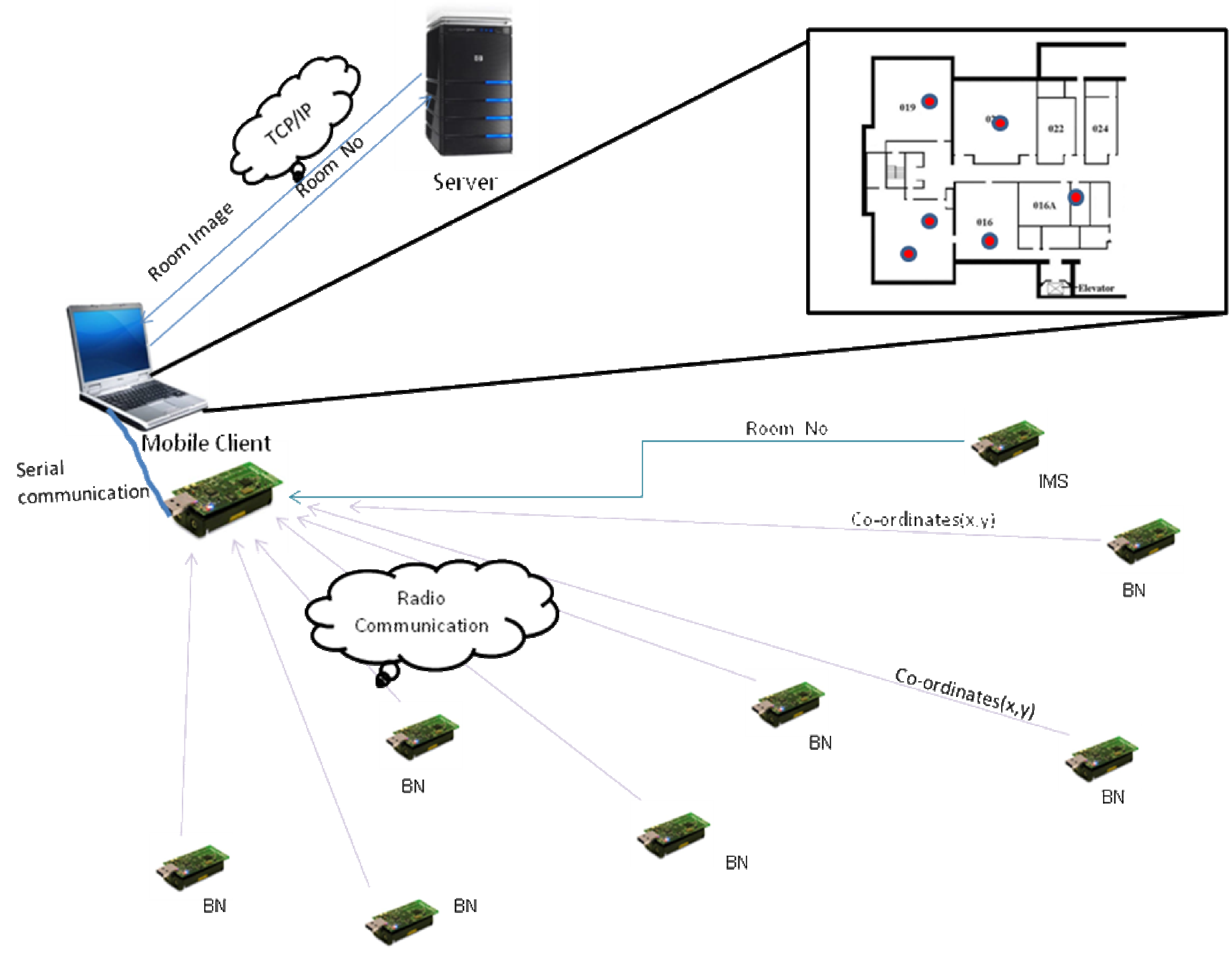


Figure 3.6 Third approach TCP/IP

CHAPTER 4 - Implementation

The basic requirement for the implementation of the project is to have Tinyos on the system being used for mote programming.

The following methods should be followed for installing Tinyos on a system:

- Install tinyos-1.11 from tinyos.net.
- Upgrade to tinyos-1.15 by using the following rpm and issuing the following Command.
- Replace the tinyos-1.x folder by the folder attached.

4.1 Motes deployment

After completing the code for the motes, it must be loaded on the Telosb mote.

The command for installing the code on Telosb is:

Make telosb install <Mote ID>

Mote ID is for identification of mote by unique source ID number.

After installing the motes with the Beacon mote code, it is placed in the room or attached with the devices. The Image Token mote is also placed somewhere in the room. It sends the room number to the base station. The base station is installed with the code and it can accept desired message formats.

For finding the port number on which the mote is connected to the system we use the following command in *cygwin*:

Motelist

This command will give comport number on which the mote is attached on the system.

Another way to find the COM port is as follows:

Right Click on My Computer->properties->Hardware->DeviceManager and scroll down.

Then the port used by the mote will be seen.

4.2 Server & Client

The server will be listening for a mobile client connection and needs to be running. It contains the map of the entire building.

The client sends the connection request to the server when it receives the room number from the Image Token mote.

4.2.1 Running Java on Ipaq

Ipaq (HP iPAQ hx2410 Pocket PC) has Windows operating system installed on it. Thus, for running java code on an Ipaq we need to have JVM (Java Virtual Machine). We have used "**mysaifu**" for the project. Here are few steps for running mysaifu on Ipaq:

- Download the mysaifu from the website
  http://www2s.biglobe.ne.jp/~dat/java/project/jvm/download_en.html
- Copy the CAB file to \My Documents.
- Tap the CAB file, Program is installed under \Program Files\mysaifu JVM folder.

Few steps for executing .jar files on IPAQ:

- Tap an icon labeled "mysaifu JVM" in "Programs" folder.
- Input following information:
  - **Class name -** Enter Java application class name or .jar file name to execute.
- When **"Execute"** button is tapped, the VM Window is shown and VM will start.
  Under the "**Options"** button the Options dialog is present.
  "Recent list" list box displays recent class names and .jar file names.
- The max heap size can also be set as per requirements.

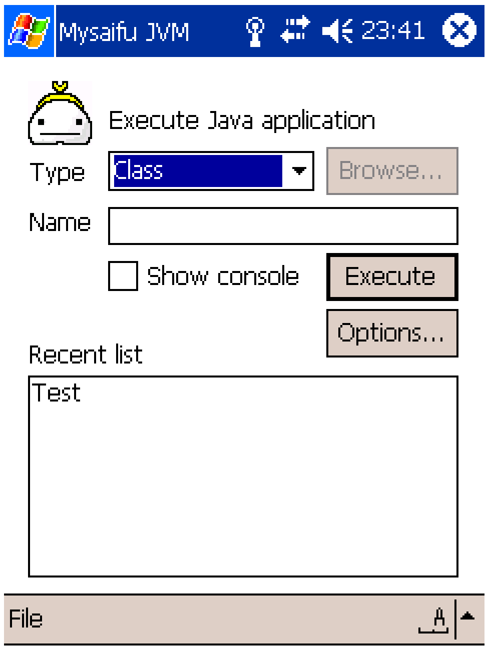
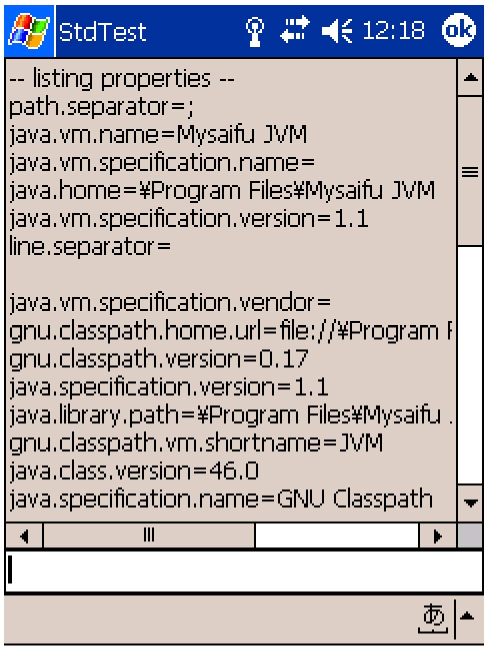


Figure 4.1 Mysaifu



```
-- listing properties --
path.separator=;
java.vm.name=Mysaifu JVM
java.vm.specification.name=
java.home=¥Program Files¥Mysaifu JVM
java.vm.specification.version=1.1
line.separator=

java.vm.specification.vendor=
gnu.classpath.home.url=file://¥Program F
gnu.classpath.version=0.17
java.specification.version=1.1
java.library.path=¥Program Files¥Mysaifu .
gnu.classpath.vm.shortname=JVM
java.class.version=46.0
java.specification.name=GNU Classpath
```

Figure 4.2 Mysaifu console

4.2.2 Connecting telosb to Ipaq

The Ipaq does not have a USB port for connecting Telosb. A USB host is needed which can help with connecting the telosb to ipaq, for example, Solar Express single port usb host. To make it work a driver must be installed which can be obtained from

http://www.lightconecorp.net/blog/?page_id=30

A driver must also be installed for Telosb on Ipaq which can be downloaded from

http://www.ftdichip.com/Drivers/VCP.htm

Figure 4.3 Solar Express Single Port USB Host card

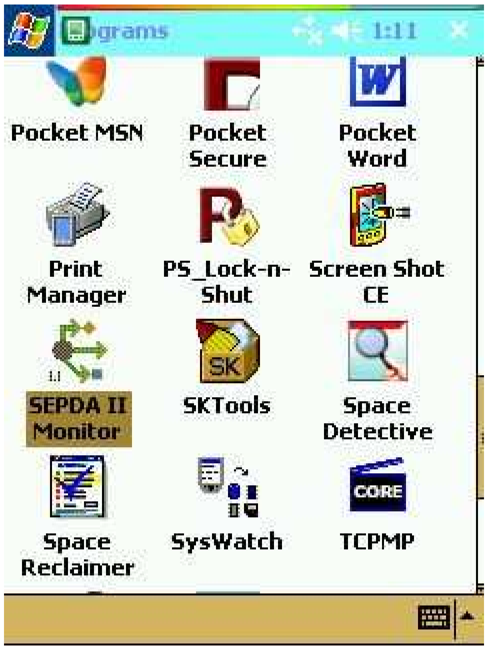
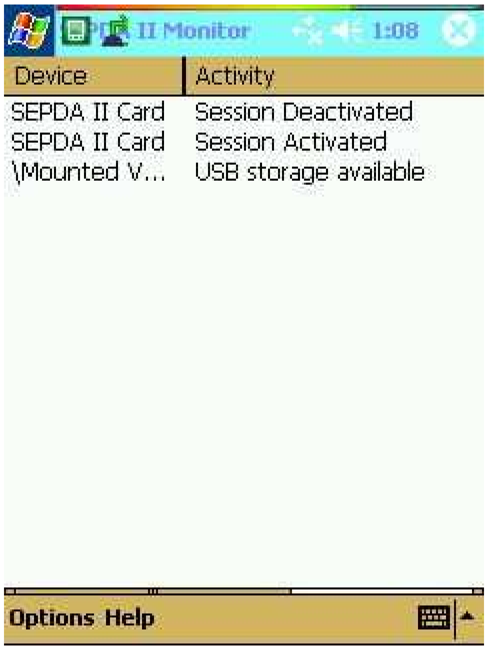


Figure 4.4 SEPDA II Monitor



Device	Activity
SEPDA II Card	Session Deactivated
SEPDA II Card	Session Activated
\Mounted V...	USB storage available

Figure 4.5 SEPDA II Monitor (Comport)

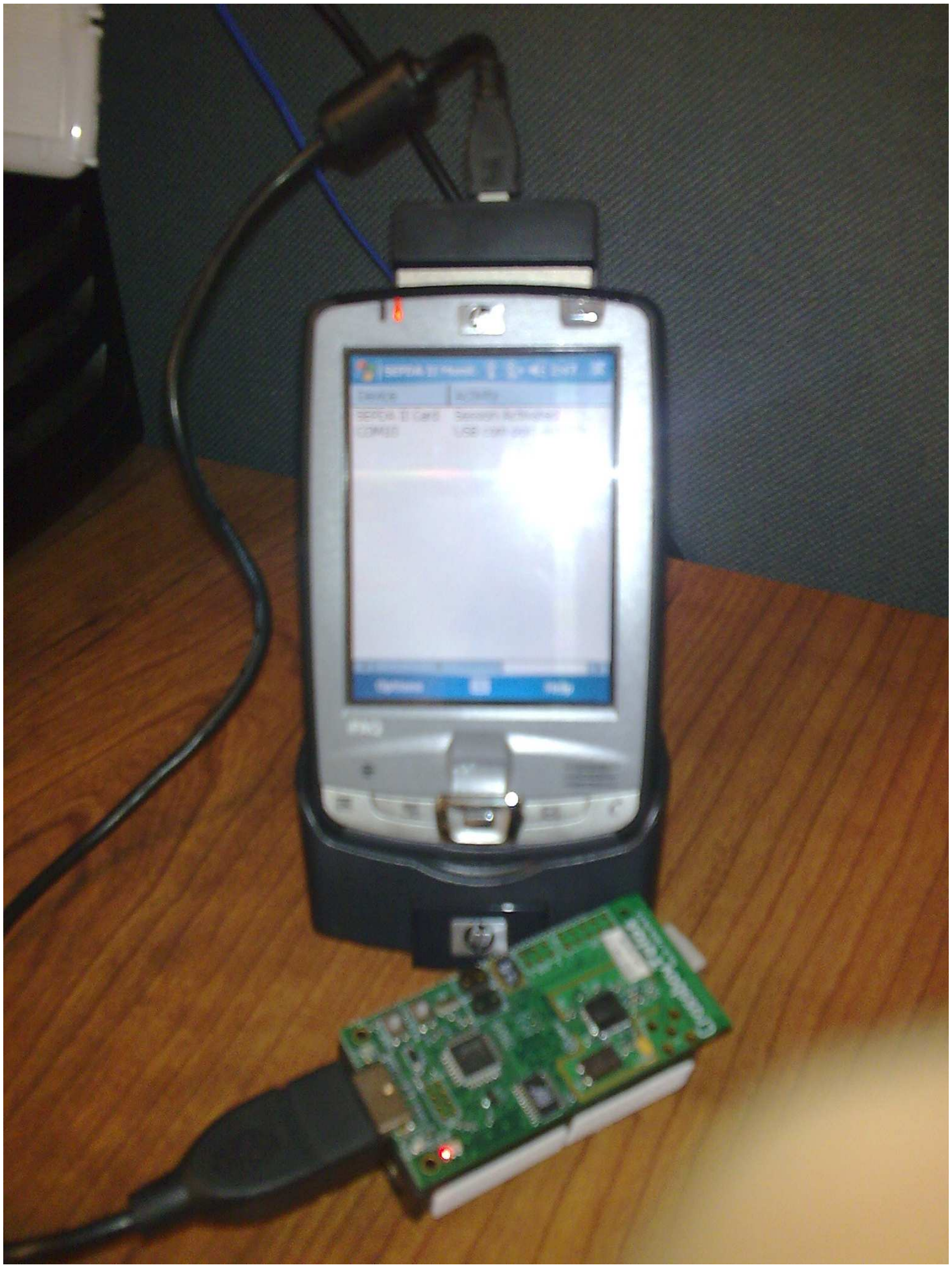


Figure 4.6 iPAQ and Telosb

CHAPTER 5 - Performance and testing

5.1 Setup for testing and output

Different scenarios were tested to evaluate the performance of the integrated system.

The setup and the tests are described below:

- Mote setup:

  The project was tested using Telosb motes in the basement of Nichols Hall. The motes were deployed in two rooms N19 and N21.

  N19 setup:

  - Five telosb motes installed with Beacon mote code was placed in Room 19.
  - One telosb mote installed with Image Token mote code is placed in Room 19.

  N21 setup:

  - Four telosb motes installed with Beacon mote code is placed in Room 21.
  - One telosb mote installed with Image Token mote code is placed in Room 21.

- Server setup:

  A laptop or any of the systems having java and internet capability is used to load server side code on it. The system has the images of room 19 and room 21 in it.

- Client setup:

  A laptop or any other device used should have JVM to compile the java code. It should also have usb port for connecting telosb to it. The system should be installed with telosb driver in it.

All the motes are started and they send their respective messages for the base station. The server code is initialized and it starts listening for the client connection. When the mobile client comes in the vicinity of the signals from the Beacon motes, it receives an Image Token variable and transfers that to the server. The server then sends the corresponding image of the room to the mobile client. The mobile client displays the image like this:

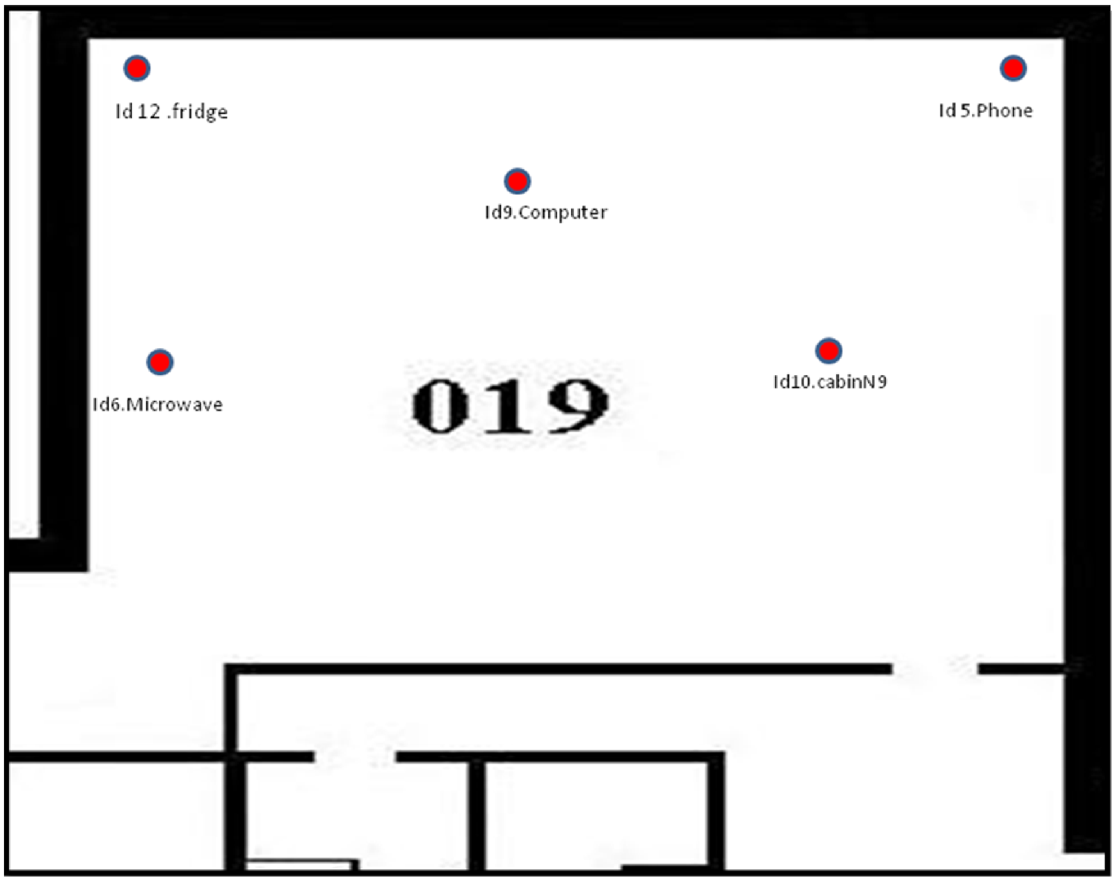


Figure 5.1 GUI display of room N19

When we move with mobile client and go near the room 21 it will display image like this:

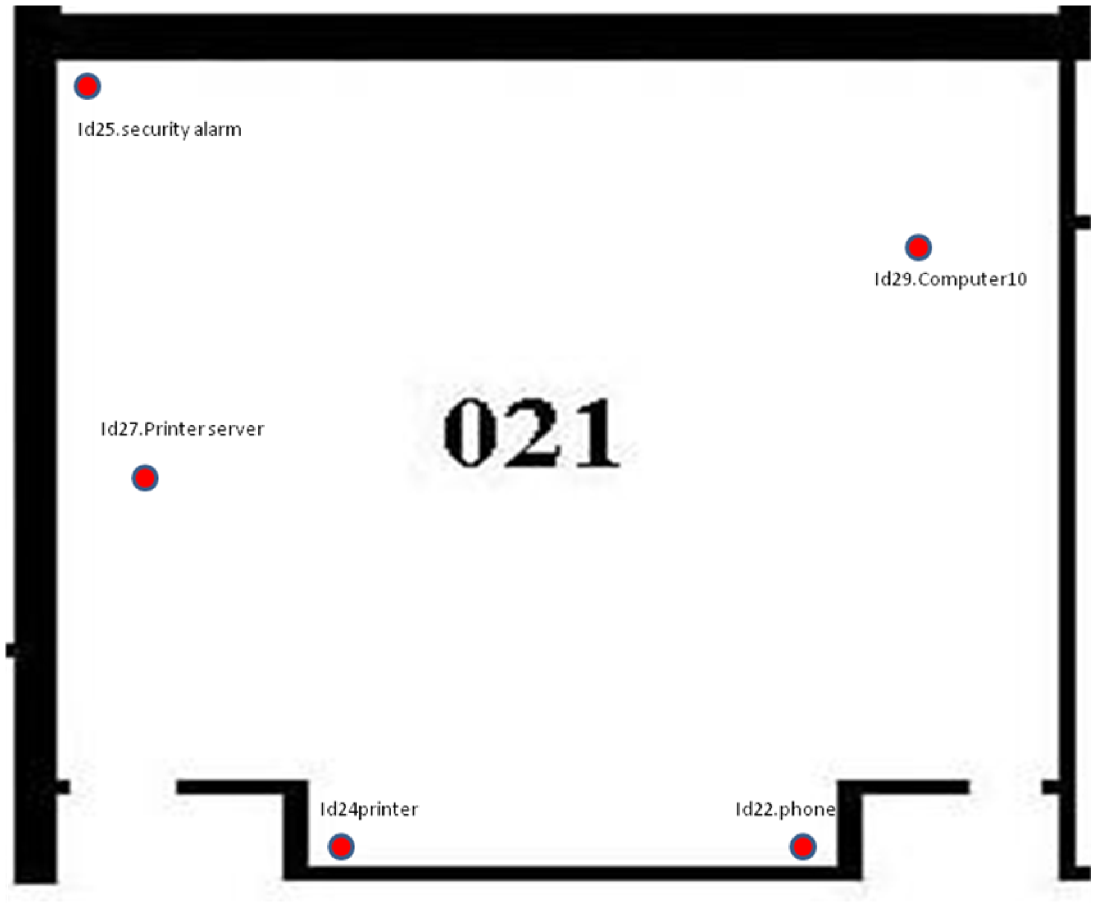


Figure 5.2 GUI display of room N21

5.2 Data collected from different test cases

The project is being tested in different ways for getting proper performance results, we have considered following test cases.

Test cases for the second design approach:

1. Time taken to display the location of mote on GUI when the image size of image is fixed and number of motes is varied (refer to table 5.1).
2. Time taken to display the location of mote on GUI when the number of motes is fixed and the size of the image is varied (refer to table 5.2).
3. Time taken to display the location of motes on GUI when the number of motes entering in the room changes (refer to table 5.3).

Test cases for the third design approach:

1. Time taken to show the location of mote on GUI when the image size of image is fixed and number of motes is varied (refer to table 5.4).
2. Time taken to show the location of mote on GUI when the number of motes is fixed and the size of the image is varied (refer to table 5.5).
3. Time taken to show the location of motes on GUI when the number of motes entering in the room changes (refer to table 5.6).

Note:-

The time shown in the performance table below is the total time taken by the mobile client to display the image with the coordinates of the beacon motes, it involve the time taken for connecting and receiving image form server and also displaying the coordinates of motes.

Observations for the second design approach

Following Table shows the observation of time taken when we vary the number of motes while keeping the image size fixed:

Number of motes	Image size considered	Time taken in second
1	800*600	6
2	800*600	7.7
3	800*600	8
4	800*600	9
5	800*600	9.4
6	800*600	9.6

Table 5.1 Time taken for localization on varying number of motes (second design approach)

Following Table shows the observation in time taken when we vary the image size while keeping the number of motes fixed:

Image size considered	Number of motes	Time taken in second
1280*800	6	9.8
800*600	6	9.6
640*400	6	9.5

Table 5.2 Time taken for localization on varying image size

Following Table shows the observation in time taken to display the new entering mote in the room while keeping the image size fixed:

Number of motes entering in a room	**Image size considered**	**Time taken in second**
1	800*600	6.5
2	800*600	7.9
3	800*600	8.8
4	800*600	9.9
5	800*600	10.2
6	800*600	10.7

Table 5.3 Time taken for localization on varying number of motes entering in a room

Observations for the third design approach

The following Table shows the observation in time taken when we vary the number of motes while keeping the image size fixed:

Number of motes	Image size considered	Time taken in second
1	800*600	4
2	800*600	4.4
3	800*600	5
4	800*600	5.2
5	800*600	5.8
6	800*600	6.2

Table 5.4 Time taken for localization on varying number of motes

The following Table shows the observation in time taken when we vary the image size while keeping the number of motes fixed:

Image size considered	Number of motes	Time taken in second
1280*800	6	6.4
800*600	6	6.2
640*400	6	6.1

Table 5.5 Time taken for localization on varying size of image

The following Table shows the observation in time taken when we vary the number of motes entering in a room while keeping the number image size fixed:

Number of motes entering in a room	**Image size considered**	**Time taken in second**
1	800*600	4.5
2	800*600	5
3	800*600	5.4
4	800*600	6.1
5	800*600	6.8
6	800*600	6.9

Table 5.6 Time taken for localization on varying number of motes entering in a room

5.3 Analysis and some more test cases

When we move with the mobile device from one room to the other, we should move in such a way that we have sufficient time in the vicinity of the sensor mote region for localization (say 7 or 8 seconds). As we can see from Tables 5.1, 5.3, 5.4, and 5.6, it requires minimum of 5 seconds to initiate the localization (to display image with motes) of motes on the mobile client. This time includes the time taken by the mobile client to connect to the server, to receive the image from the server and to display coordinates of motes on it. The experiment is tested in the basement. As shown in Figures 5.1 and 5.2 the path for the test case is:

Staircase ---->> corridor 1 ---->> N19 ---->> corridor 2 ---->> N21

We have the main mote setup in rooms N19 and N21. For checking the system more efficiently there are some motes placed in the corridor also. Total time taken to cover the path is 26 seconds.

Steps involved in running the project:

1. Mobile client comes in the vicinity of sensor region.
2. Mobile client receives the image token variable from Image token mote.
3. Mobile client send that token to the server.
4. Server responds to the client with the image of the room.
5. Once the mobile client gets the image it starts receiving the coordinates from other motes (beacon motes).
6. Mobile client displays the coordinates of the beacon motes on the room map.

Following table shows the amount of time taken in displaying the individual sensor region with the mote location on mobile client:

Sensor area	**Number of motes**	**Time taken in seconds**
Corridor 1	2	6
N19	6	7.4
Corridor 2	2	4
N21	5	6.3

Table 5.7 Time taken for localization of different sensor areas

The table below shows the approximate average time for each individual process while running the experiment.

Individual processes for running the project	**Time taken in second**
Connecting with the server	2.2
Receiving and loading image from server/image	3
Displaying coordinates on GUI/coordinate	.3
Receiving messages from image token mote and beacon motes/message	.3

Table 5.8 Time taken by individual processes

5.4 Using the system to locate a device in vicinity

Experiment: Finding Microwave device in Nichols basement.

We start our program on the mobile client and start moving the mobile client in the basement, the base station attached with the mobile client will look for the radio signals from other motes. As we pass by (make sure to walk slowly enough so that you could have 7 to 8 seconds while crossing) the corridor the base station receive image token variable from image token mote and sends that variable to the server. In response, the server sends the corridor image to the client, client loads the image and start displaying the coordinates of the beacon motes along with the device ID they are attached to. After displaying the coordinates on the corridor map, we found out no Microwave device is placed in the corridor. We continue to move in our path and reached in hall N19, mobile client receive radio signal from image token mote and send it to server. In response server sends the image of room N19 to the mobile client, mobile client load that image on GUI and start displaying the coordinates of the beacon motes along with the device ID they are attached to. After looking at the GUI we found out that microwave (with device ID) is found on GUI image as shown in figure 5.1.

CHAPTER 6 - Conclusion

The system has been tested with three different approaches and it was found that the third approach with the TCP/IP server socket works best for the problem provided. The Bluetooth approach is not reliable in message transfer as it often loses connection and has a problem with the initial connection setup between devices. The second approach in which there are two different server socket connections between devices for transferring image and for transferring mote messages also has a problem in its design structure. The server should always be in the vicinity of the motes for transferring messages from the motes to the mobile client. The third approach in which there is only one server socket connection from server to client for both, image transfer and mote messages, the mobile client can directly talk with the motes with the help of the base station connected to the mobile client. This approach does not have a constraint of placing the server in the vicinity of the motes and also the connection with TCP/IP is much more reliable than the Bluetooth connection.

CHAPTER 7 - Future work

The project requires offline installation of Beacon motes. To avoid this we need an adhoc mechanism for rapidly installing Beacon mote system. The project is for static localization in sensor network. We need an approach which can also work dynamically but for that we need to come up with an idea which should be accurate with locating the positions of motes.

If there is any change in IP then we need to change the IP in the code again to make it work with the server socket approach. We need to come up with an approach which can deal with change in IP of the system.

References

[1] Tinyos concepts and Nesc Programming language
http://www.tinyos.net

[2] Tiynos and Nesc programming problems
http://mail.millennium.berkeley.edu/pipermail/tinyos-help

[3] Mote Track
http://www.eecs.harvard.edu/~konrad/projects/motetrack/

[4] Mysaifu JVM
http://www2s.biglobe.ne.jp/~dat/java/project/jvm/index_en.html

[5] Solar PDA Blog
http://www.lightconecorp.net/blog/?page_id=2

[6] Sending & Receiving Messages in TinyOS
http://www.ece.utk.edu/~xwang/ece455/Robert.pdf

[7] Robust location tracking using a dual layer particle filter
http://portal.acm.org/citation.cfm?id=1238460