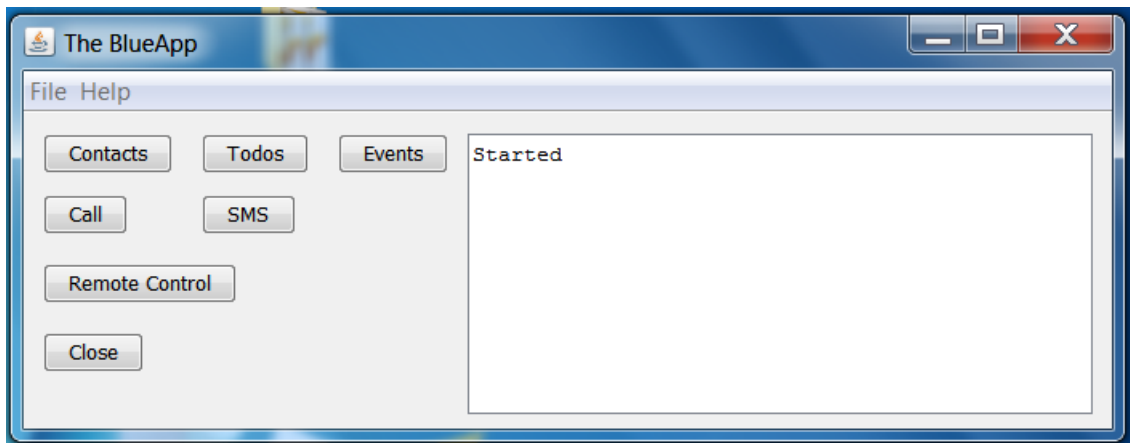


The Problem Statement



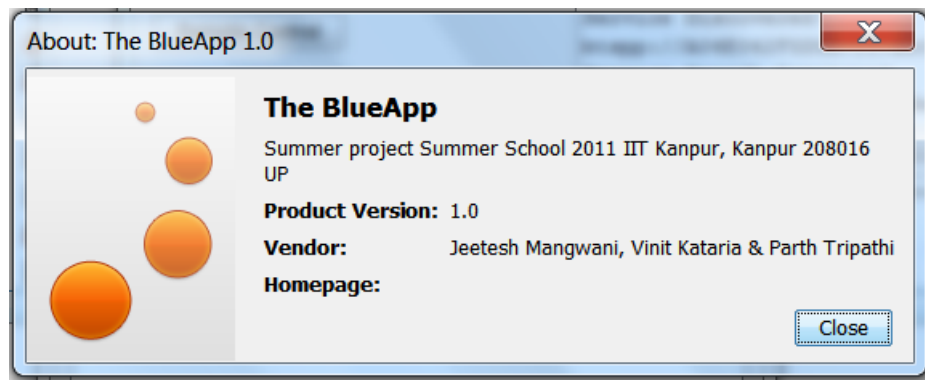
The project aimed to create an application in Windows/Linux that facilitates performing the following operations on the cell connected to the Bluetooth port of the system:

- 1) Managing Contacts
 - a. Listing all the contacts
 - b. Creating a new contact
 - c. Editing
 - d. Deleting
- 2) Managing To-Dos
 - a. Listing all the to-do
 - b. Creating a new to-do
 - c. Editing
 - d. Deleting
- 3) Managing Events/Meetings
 - a. Listing all the event
 - b. Creating a new event
 - c. Editing
 - d. Deleting
- 4) Controlling the system remotely
 - a. Initiating applications like-
 1. Task Manager
 2. Notepad
 3. MS Paint
 4. Windows Explorer
 5. Calculator
 - b. Going to previous/next slide in a PowerPoint presentation
 - c. Going to previous/next picture in a picture viewer

- d. Closing the active window
- 5) Managing SMSs
 - a. Sending an SMS
 - b. Receiving an SMS*
 - c. Browsing Inbox, Sent messages etc *
- 6) Managing mass memory#
 - a. Transferring files#
 - b. Renaming files#
 - c. Cutting, Copying and Pasting files#
- 7) Managing calls
 - a. Making a call
 - b. Receiving a call*
 - c. Browsing call history*

*These features were not possible to be added due to constraints set by mobile OS and J2ME technology

#These features could not be added due to shortage of time

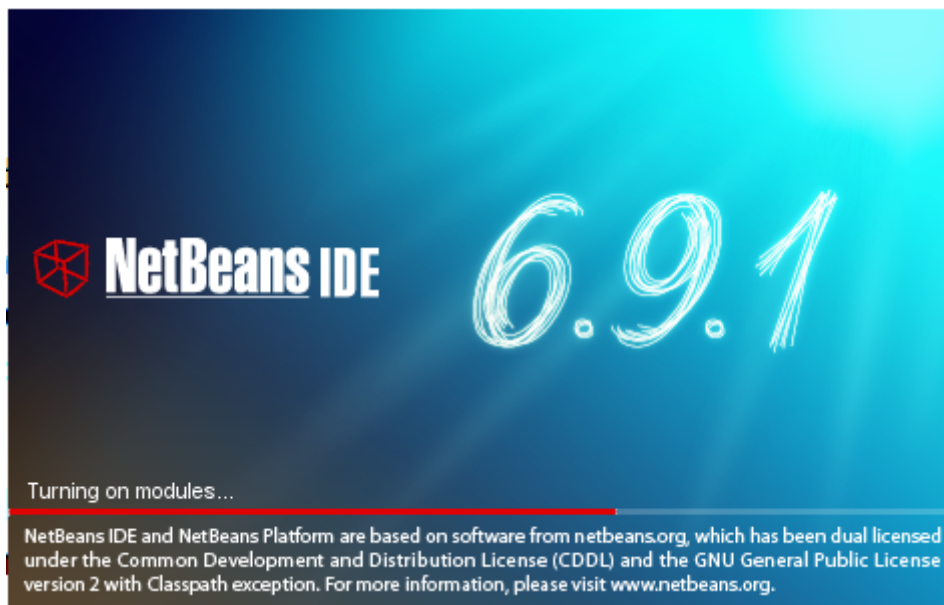


The Choice of platform

We considered the following facts:

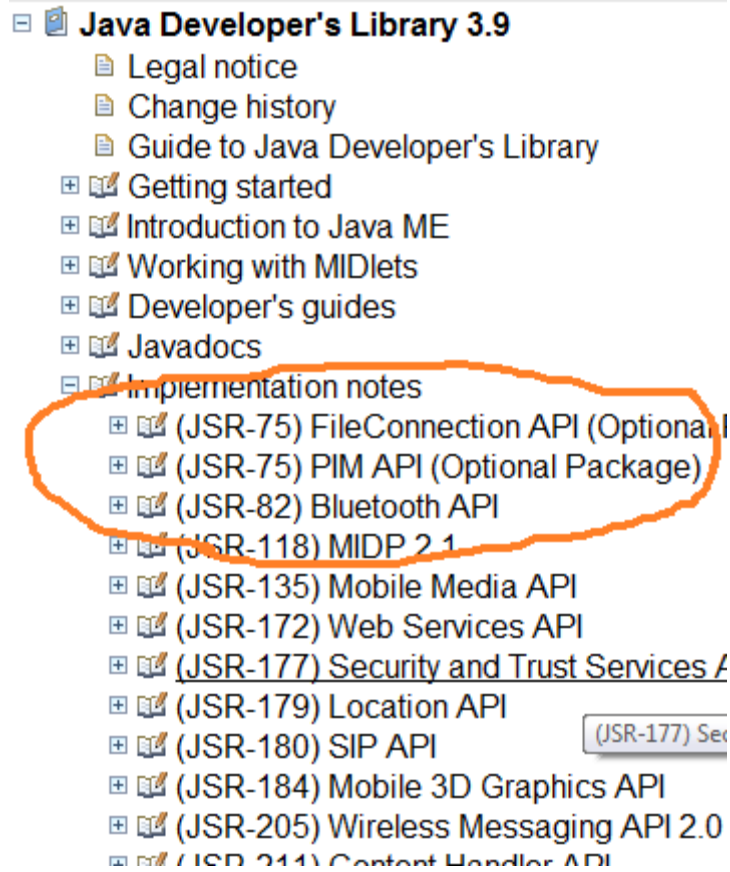
- a. Widespread support for J2ME technology in handsets
- b. Unparalleled Developer-support at forum.nokia.com
- c. Availability of Symbian S60 5th edition on our two Nokia 5233 handsets

And decided to code in NetBeans 6.9.1 IDE integrated with Symbian S60 5th edition Emulator platform as the cell phone simulator.



APIs used

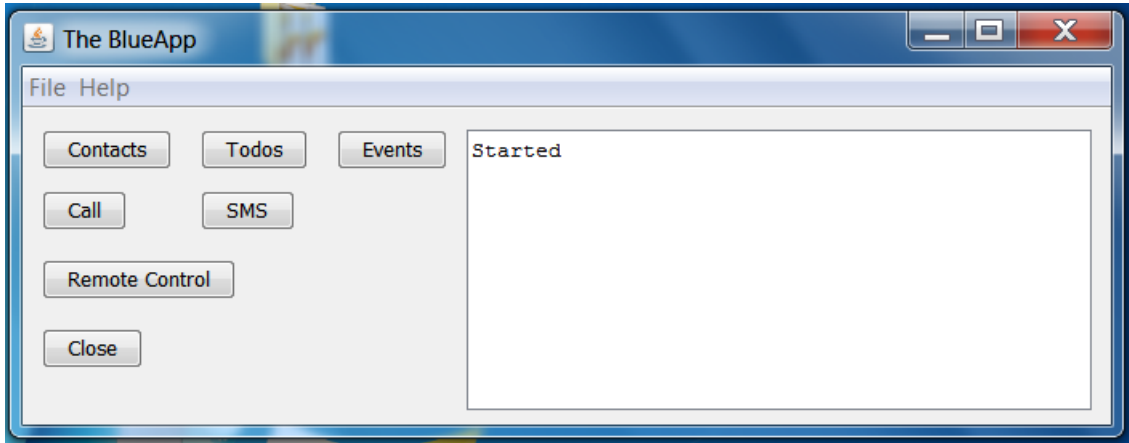
1. JSR-82, the Java Bluetooth API
2. JSR-75, the PIM (Personal Information Management) and File Connection API



The Java Developer documentation support at forum.nokia.com

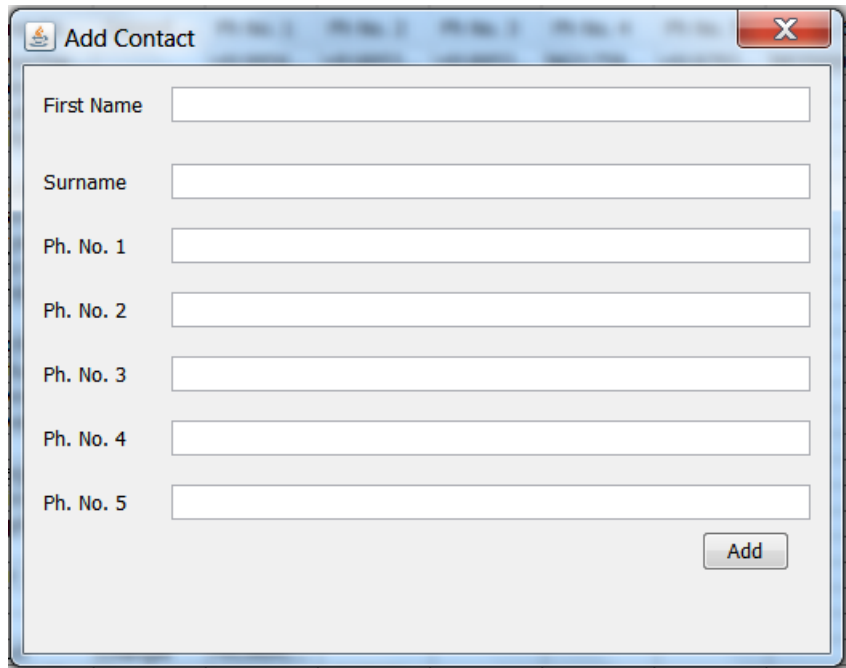
The Code Structure

The Computer-side Code



1. **SingleFrameApplication** TheBlueApp
 - a. startup()
 - i. calls findClient() and menuHandler()
 - b. findClient()
 - i. Discovers Bluetooth devices in neighbourhood
 - ii. Finds the service corresponding to this application
 - c. menuHandler()
 - i. Initiates the welcome screen and the main session
 - ii. Keeps calling manageContacts(), manageToDos(), managerEvents(), managerCalls(), manageSMSs(), manageRemote() and closeApp() depending upon user input
 - d. manageContacts()
 - i. starts the **JFrame** contactFrame and its main thread
 - e. manageToDos()
 - i. starts the **JFrame** contactFrame and its main thread
 - f. manageEvents()
 - i. starts the **JFrame** contactFrame and its main thread
 - g. manageSMSs()
 - i. starts the **JFrame** contactFrame and its main thread
 - h. manageCalls()
 - i. starts the **JFrame** contactFrame and its main thread

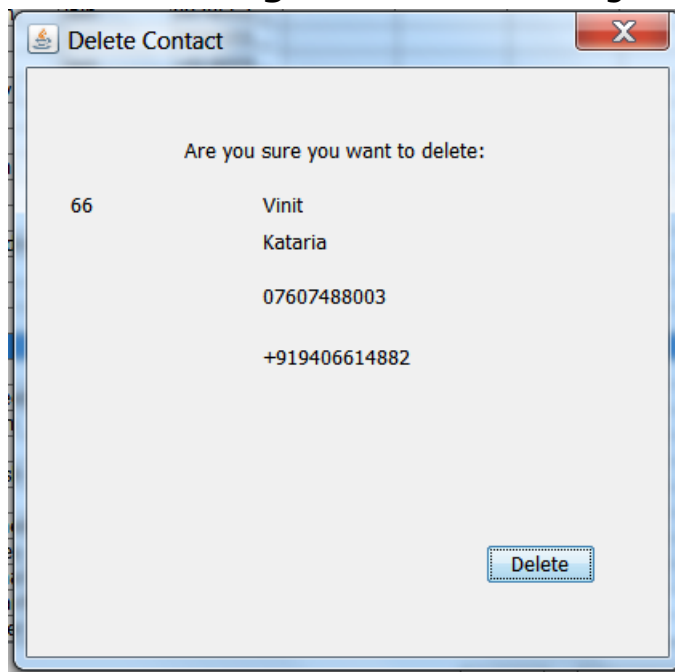
- i. manageRemote()
 - i. starts the **JFrame** contactFrame and its main thread
 - j. closeApp()
 - i. ends the main session
2. **JFrame** contactFrame
- a. starts the thread that manages operations related to contacts
 - b. initiates **JDialog** addContactDialog



The screenshot shows a Java Swing dialog box titled "Add Contact". It features a standard window title bar with a close button (X). The dialog contains the following elements:

- First Name:
- Surname:
- Ph. No. 1:
- Ph. No. 2:
- Ph. No. 3:
- Ph. No. 4:
- Ph. No. 5:
- At the bottom right, there is an "Add" button.

- c. initiates **JDialog** deleteContactDialog



The screenshot shows a Java Swing dialog box titled "Delete Contact". It features a standard window title bar with a close button (X). The dialog contains the following elements:

- Text: "Are you sure you want to delete:"
- Contact details:
 - 66
 - Vinit
 - Kataria
 - 07607488003
 - +919406614882
- At the bottom right, there is a "Delete" button.

- d. initiates **JDialog** modifyContactDialog

Modify Contact

First name

Surname

Ph No. 1

Ph No 2

Ph No 3

Ph No 4

Ph No 5

3. **JFrame** ToDoFrame

- a. starts the thread that manages operations related to To Dos
- b. initiates **JDialog** addToDoDialog

Add ToDo

Summary

Due Date

Hrs Day

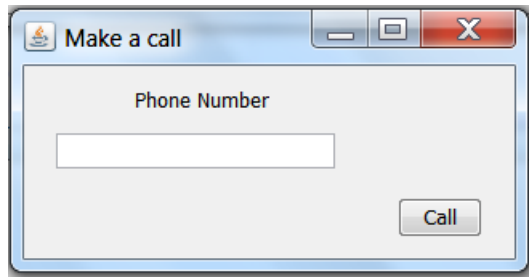
Min Month

Sec Year

- c. initiates **JDialog** deleteToDoDialog
- d. initiates **JDialog** modifyToDoDialog

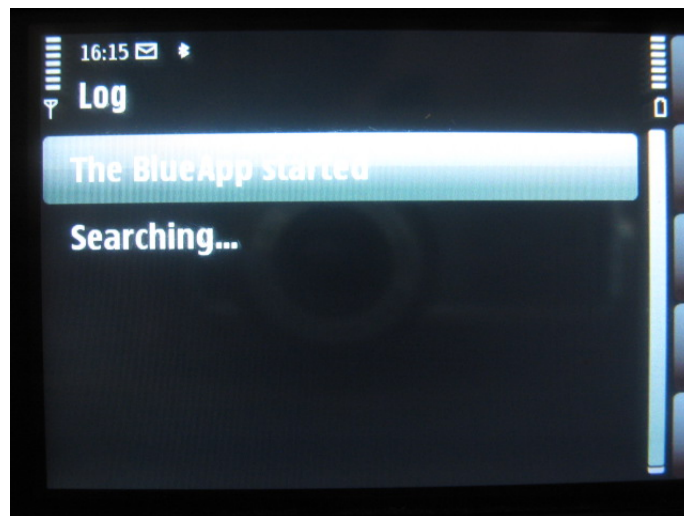
4. **JFrame** contactFrame

- a. starts the thread that manages operations related to Events
 - b. initiates **JDialog** addEventDialog
 - c. initiates **JDialog** deleteEventDialog
 - d. initiates **JDialog** modifyEventDialog
5. **JFrame** callFrame
- a. starts the thread that initiates a call



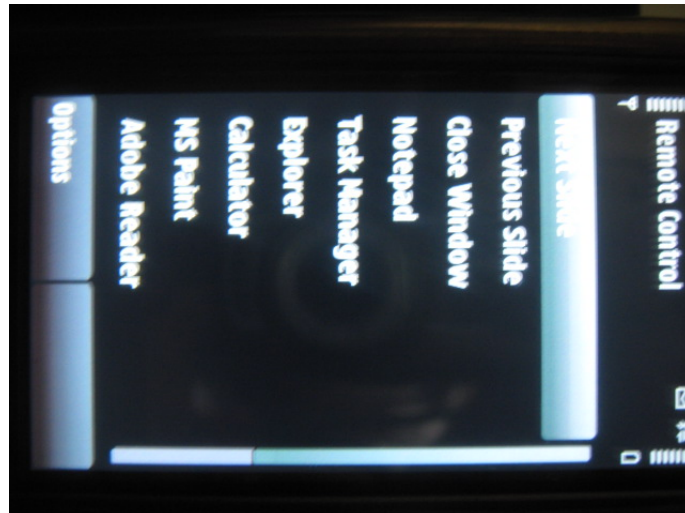
6. **JFrame** smsFrame
- a. starts the thread that sends sms

The Client-side Code



1. **MIDlet** BTServerCell
- a. Instantiates **ManageContacts** contactManager and starts its main thread t
 - b. Instantiates **ManageToDos** toDoManager and starts its main thread t
 - c. Instantiates **ManageEvents** eventManager and starts its main thread t
 - d. Instantiates **ManageSMSs** smsManager and starts its

- main thread t
- e. Instantiates **ManageCalls** callManager and starts its main thread t
 - f. Instantiates **ManageRemote** remoteManager and starts its main thread t



- 2. class **ManageContacts**
 - a. **Runnable** t (manages the Bluetooth communication in case of "Managing contacts" option)
- 3. class **ManageToDos**
 - a. **Runnable** t (manages the Bluetooth communication in case of "Managing ToDos" option)
- 4. class **ManageEvents**
 - a. **Runnable** t (manages the Bluetooth communication in case of "Managing Events" option)
- 5. class **ManageSMSs**
 - a. **Runnable** t (manages the Bluetooth communication in case of "Sending SMS" option)
- 6. class **ManageCalls**
 - a. **Runnable** t (manages the Bluetooth communication in case of "Managing Calls" option)
- 7. class **ManageRemote**
 - a. **Runnable** t (manages the Bluetooth communication in case of "Remote Control" option)



Log



Call Manager Started

07607488003

Calling: 07607488003

Success in calling

Back in main menu

Problems encountered and their solutions

1. Integrating NetBeans IDE with S60 Emulator

Sometimes automatic facility of NetBeans integration with S60 Emulator fails. In such cases, one has to manually integrate them, clearly specifying the S60 folders to look into for the missing classes, javadocs and libraries.

2. Bluetooth Service not discovered

In the initial stages of the application development part, we were facing problems in the discoverability of Bluetooth devices, only until we realised that the problem lies in the fact that the default setting of Bluetooth devices switches off the service discoverability after about 10 seconds. Therefore, you must run the application immediately after you switch on the Bluetooth adaptor, that is, the devices won't be discovered in the application if the Bluetooth adaptor had been On for a reasonable time!

3. AWT package Event-dispatching thread problem

The windows and dialog boxes could not be seen at places and times we expected them to. Wildly searching on the java developer website for days, something called "Event Dispatch Thread" and "Thread safety" came to our notice.

And, it solved our problem. For details go to <http://java.sun.com/products/jfc/tsc/articles/threads/threads1.html>.

4. Library reference problem

The application works fine when run using an IDE like the NetBeans 6.9.1 IDE, but there are still some issues when tried to run using only a Java Virtual Machine. There comes an error regarding the library reference not found. We are still searching and working on this issue but soon this will be fixed too.

5. Application descriptor error

When making the MIDlets in the NetBeans 6.9.1 IDE, one of the major problems faced during building the MIDlet was an error stating that there is no MIDlet defined. After a lot of searching on the issue, it was found that NetBeans sometimes does not define/locate the MIDlet by itself and it has to be added into the Application Descriptor in the Properties of the application.

Further improvements

a. Extending the "Remote Control" feature

The Remote Control feature of our application can be extended to perform many more tasks in the computer (using the cell phone) as per the requirements, which have not been included in our application because of lack of time.

b. If only we used Symbian C++ instead of Java ME

The reason for us choosing Java ME as a language in our application was basically to give it mobility over many platforms and making it more compatible. For this very reason, we had to adhere to the limitations of Java ME in mobile system related tasks like limited support by the mobile OS to SMS related tasks (extracting messages from inbox, etc), calling/receiving tasks ,etc.

Some of these limitations can be resolved if Symbian C++ is used as the programming language for the application (as it has extended support by the mobile OS), which although limits the mobility of the application to very limited number of cell phones, but gives more accessibility into the system to the application.

c. Extending other features of the application

Other features, including important and much useful ones like File Transfer, can be added further to our application. This would add to the usefulness of the multi-utility application. Moreover, many other commands and mouse pointer control using the cell phone can also be added, which could not be added due to lack of time.

Uses

The application can be of great utility in performing various tasks in the computer using the cell phone and vice-versa.

The tasks like managing a large number of contacts, to dos and events can be **really cumbersome when done on a cell phone**, while it is lot more easier to manage these on a personal computer. The computer-side application can be really useful in this case as it displays (lists) all the contacts, to dos and events in a **tabular form** and allows easy management of these.

The application can also be used to initiate phone calls to a phone number entered by the user, using the computer-side application.

Another important use of the application is to send SMSs. We all know that be really **'irritating' in a cell phone** even if it has a QWERTY keypad, here, the application can be used for typing SMSs in your computer and sending them.

One of the most important feature of the application is that it can be used to use **cell phone as a 'remote-control'** . The remote-control feature includes using cell phone to open various other applications/ utilities on the computer like notepad, calculator, task manager, MS Paint, windows explorer (and many more can be added), and that too with just a click of a button on the cell phone. The cell phone can also be used to close the active window.

A good use of the cell phone as a remote control is to move to the next/previous slide during a powerpoint presentation or to move to the next/previous picture in a picture viewer.

The most important feature/biggest plus point of the application is its mobility over various platforms!

Well, further extensions could have been made in the application if there would have been more time .

The application can be used commercially too as a multi-utility Bluetooth application.

Summarised write-up

The BlueApp is a small cellphone multi-utility that enables you to easily manage, browse, add, edit and delete the contacts & reminders of your cell phone on the PC screen. In addition, you can also initiate phone calls and send SMSs.

Moreover, you can use your cell phone as a device to control your PC remotely viz. Moving between slides during a Powerpoint presentation, Moving between tracks on a music player, starting Notepad, MS Paint etc, Moving the mouse pointer on the PC screen.

The application has been developed on NetBeans 6.9.1 IDE after integrating it with Symbian S60 5th edition SDK. We used JSR-82 (the Java Bluetooth API) and JSR-75 (the PIM and File Connection API) for J2ME-coded cell phone-side application, alias MIDlet.

We used Blue Cove as the Bluetooth library for J2SE-coded PC-side application. The GUI was constructed using Java Swing Framework. The Nokia Forum (forum.nokia.com) and The Really Big Index (<http://download.oracle.com/javase/tutorial/reallybigindex.html>) were really helpful when we were stuck at places.

As more cell phones start supporting the Telephony API JSR-253, we can add features like receiving calls on cell phone. Inserting the file management features will add further to the utility. All in all, it was a breathtaking firsthand experience at professional application development marking our first step into the computer engineering world.