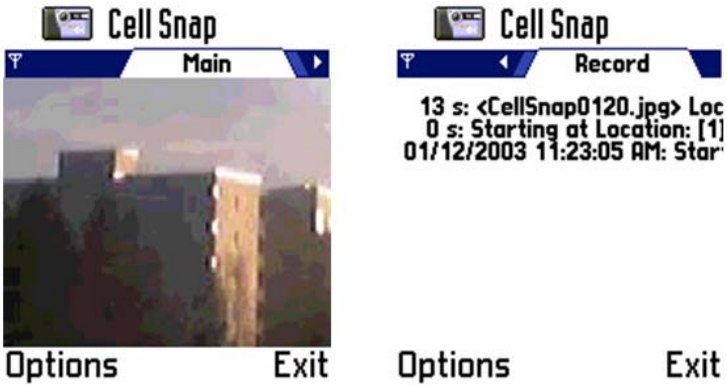
Department of Computer Science and Engineering

The Chinese University of Hong Kong

2003-2004 Final Year Project

Location-based Services using GSM Cell Information over Symbian OS

LYU0301



Supervised by Professor Michael LYU Rung Tsong

Group members: Mok Ming Fai
Lee Kwok Chau

Prepared by Mok Ming Fai (01713554)

Abstract

This project is subjected to the final year project LYU0301. The project title is “Location-based Services using GSM Cell Information over Symbian OS”.

This report contains ten chapters. They include the introduction of this project, the introduction of Symbian OS, some basic GSM technologies, currently existing LBS solutions, location-based service using location area identifier (LAI) and cell identifier (CI), LBS in the 2-dimensional space, LBS in the 1-dimensional space, the middleware architecture and its component.

Our final year project tries to make use of LAI and CI to provide location-based service in 2-dimensional and the 1-dimensional space using simple hardware which is going to be widely distributed in the near future - mobile phones with Symbian OS.

We also created a middleware for developing location-based application using our approach in LBS. This middleware includes a set of APIs and tool kits.

This report also includes two test on the middleware developed. We tried to remake MTR Traveller and create a new application, CU Campus Bus Route, using the middleware.

We performed investigations and experiments for all the possible ways to accomplish this project. This report includes the descriptions and results of all the experiments done. Programs developed are also explained.

Chapter 1 to chapter 7 are all work done in the first semester while chapter 8 to 16 belongs to those work done in the second semester.

Table of Content

1. Introduction

1.1 Motivation

With the rapid development of information technology, the need of various types of information service emerges. Among them the location sensitive applications are of the utmost importance. There are quite a number of researches being done in this field recently to look into a way to provide location-based service to the general public.

Knowing the location where one is at can be of great help in daily life. For example, one can find out nearby restaurants or fast food chain shops for lunch; one can also find out where to take a bus or change for MTR. There is really a need of location-based service that can be used by the general citizens. Therefore we would like to try to have a study in this field.

1.2 Project Objective

There are currently a number of existing location-based services (LBS) available. However, they require some special hardware in order to provide services. This make the service not easily distributed among the general citizens. We would like to work on a system that requires the minimum special hardware requirement and system overhead while providing LBS.

Our approach is to use the location area identifier (LAI) and cell identifier (CI) to approximate our current location. These information can be retrieved from the signal emitted by GSM base station. However, retrieving these information is not easy, since most mobile phone does not support this operation. This is why we turn to use mobile phones that utilize Symbian OS, which is power enough for us to carry out this operation.

With the help of Symbian OS, we tried to implement an LBS that only requires a mobile phone to operate. In the following chapters, we will try to explain the basic knowledge of Symbian OS, GSM Technologies first. After that, we will describe and explain our work in this semester.

2. The Symbian OS

2.1 The Rise of Symbian

Symbian, founded in June 1988, is a software licensing company jointly owned by several wireless industry leaders including Ericsson, Nokia, Panasonic, Psion, Samsung Electronics, Siemens and Sony Ericsson. It is a private independent company developing advanced, open, standard operating system for data-enabled mobile phones. Its product, Symbian OS (Symbian Operating System), is becoming popular in the mobile phone market.

There are currently quite a number of mobile phone using Symbian OS. They include:

- Nokia 6600, 7650, 3650, N-Gage and 9210 Communicator
- Sony Ericsson P800, P900
- Motorola A920
- Fujitsu F2051, F2102V

There will be more mobile phone producers turning to this new operating system. It is announced that the following mobile phones, which use Symbian OS, will be released very soon.

- Samsung SGH-D700
- Siemens SX1
- Sendo X
- BenQ P30

2.2 Operating System for Mobile Devices

Symbian OS is a robust multi-tasking operating system specially designed for real-world wireless environments and the constraints of mobile phones like limited amount of memory and limited speed of CPU. It is 32-bit, little-endian operating system working with ARM architecture chips with V4 instruction set or higher. Supported platforms include PrimeXSys platform from ARM, the StrongARM, Xscale architectures

form Intel, the OMAP platform from Texas Instruments and the Dragonball platform from Motorola.

Symbian OS not only works on mobile phone, but also work on any mobile devices that satisfy the requirement of the operating system. This operating system contains an extensive and rich collection of libraries and for implementing many industry standards.

Industry Standards	Examples
Networking	TCP/IP, PPP, TSL, SSL, IPSec, FTP
Communications	Bluetooth, IrDA, Obex
Security	DES, RSA, DSA, DH
Messaging	POP3, IMAP4, SMTP, SMS, BIO
Browsing	HTML, HTTPS, WAP, WML
Telephony	GSM, GPRS, fax
Multimedia	WAV, AU, WVE, JPEG, BMP, MBM, GIF

Table 2.1: Industry Standards Supported by Symbian OS

Symbian OS is so powerful that, besides supporting a number of industry standards, it also support a lot of useful functions in various field for developers to develop application for both computation and telephony.

For example, Symbian OS allows developers to retrieve cell information and information of telcos via the frame sent out from base station; There is a DBMS in the phone for better data storage efficiency; There is also APIs for developers to use the camera.

All these strength are the reasons for us to turn to use Symbian OS as our working platform in our FYP.

2.3 Characteristics of Symbian OS

Symbian OS has a number of characteristics that make it suitable for mobile computing and better against other operating system.

1. **Integrated Multi-Mode Mobile Telephony**
   Symbian OS integrates the power of computing with mobile telephony, supporting complex computations, advanced data service and telephony service on one single mobile machine

2. **Open Application Environment**
   Applications written in different programming languages, like C++ and Java, can be deployed onto Symbian OS

3. **Open Standards and Interoperability**
   Symbian OS provides a core set of APIs that is shared by all Symbian OS phone. Thus, a program written using the core set of APIs can be used in many Symbian OS phones without the need of modifying the source and recompiling.

4. **Multi-tasking**
   Symbian OS fully implements multi-tasking and threading. Its special feature active object allows program require multi-threading to be written with only one thread with several active objects.

5. **Fully Object-oriented**
   Symbian OS is designed completely using object-oriented techniques. It also allows program to be written using OO technique.

6. **Flexible User Interface**
   Graphical user interface in Symbian OS is flexible, users can design their own user interface easily. For example, Nokia developed its own specific GUI called “Avkon” for use in its mobile phone.

2.4 System Structure of the Symbian OS

The system structure of the Symbian OS is quite straight forward. The operating system and its applications can be divided into 4 types of components. They are kernel, engines, servers and applications.

The kernel is used to manage the hardware inside the system. They may include RAM, display device and keypad etc. In Symbian OS, hardware devices can only be accessed by privileged APIs, which in turn can only be executed by the kernel. Other applications have to access these system resources by calling the kernel APIs.

A server is a program that do not have graphical user interface. It manages resources that may be needed by clients. Examples of server include file server, which helps to manage the operations of the system, and window server, which helps to manage the display area on the screen.

An engine is part of an application that only manipulates data and do not interact with the user. An engine helps to separate the program core from the part that manipulates the GUI. This makes the program more flexible. For example if we want to change the interface, we don't need to have any modification on the core engine.

An application is just a program that interacts with users with an interface. It may need to access to servers for certain resources.

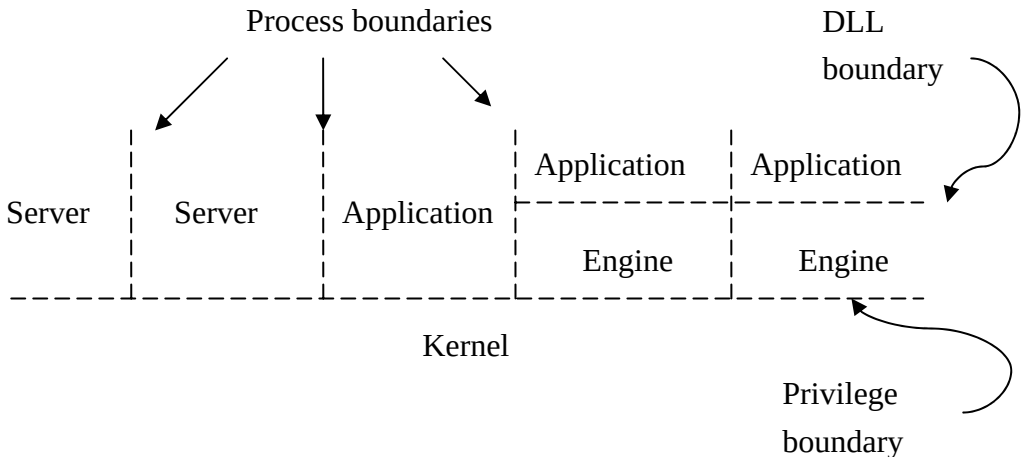


Figure 2.1: The components of the Symbian OS and its applications

2.5 Special Features of Symbian OS

There are a number of special implementation on the Symbian OS that make it a robust and suitable operating system for mobile devices with limited memory and computing powers. These special features include Descriptors, Cleanup Stack, Two-phase Construction, Active Object.

2.5.1 Descriptors

2.5.1.1 Descriptor Overview

In C, strings processing is inconvenient. Strings are NULL-terminated, so programmers have to allocate the extra byte to accommodate the terminator. A slightly wrong in arithmetic while manipulating the string may cause undesired memory overwrites. Therefore using strings in C may easily create bugs.

In Symbian OS, descriptors are used instead for string handling. Descriptor provides safe and consistent mechanism in dealing with strings and even binary data. Descriptor object maintains pointer and length information to describe data. It is used to access and manipulate the data. Descriptor object provides safe access and manipulation to the data it refers. Attempts to access memory outside the data area represented by the descriptor object are caught. In the current version of Symbian OS, two different sizes of descriptors are provided. They include 8-bit version and 16-bit version.

2.5.1.2 Classification of Descriptors

We can divide descriptors into different types from different point of view. In term of the location where the data and the descriptor object exist, we may classify descriptors into Buffer Descriptor, Heap Descriptor and Pointer Descriptor. In term of the modifiability, we may classify descriptors into modifiable and non-modifiable.

1. **Non-modifiable Descriptor**

   Non-modifiable descriptor can be accessed but cannot be changed (except complete replace). Examples are TBufC, TPtrC

and HBufC. All descriptor class ends with a capital letter ‘C’ indicates that the descriptor object is non-modifiable. The length information stored in non-modifiable descriptor is the current data length.

2. **Modifiable Descriptor**
   Modifiable descriptor can both be accessed and changed. They include TBuf, TPtr. Besides the current data length, modifiable descriptor contains an extra length information, the maximum length of data that can be represented by the descriptor. This extra information is needed so as to make changes to the data within.

We may also classify descriptors in term of their structure:

1. **Buffer Descriptor**
   Data is part of the descriptor object and the descriptor object lives in the program stack.

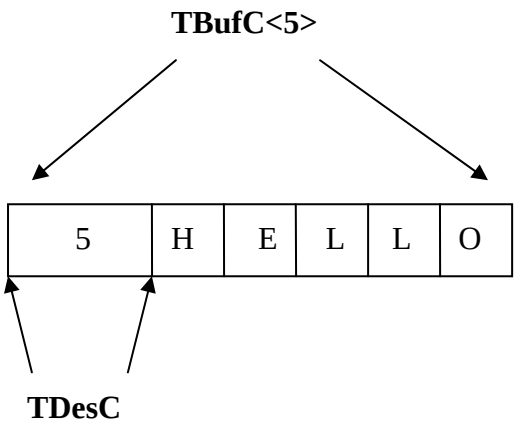


Figure 2.2: Representation of a Buffer Descriptor

2. **Heap Descriptor**
   Data is part of the descriptor object and the descriptor object lives in the heap.

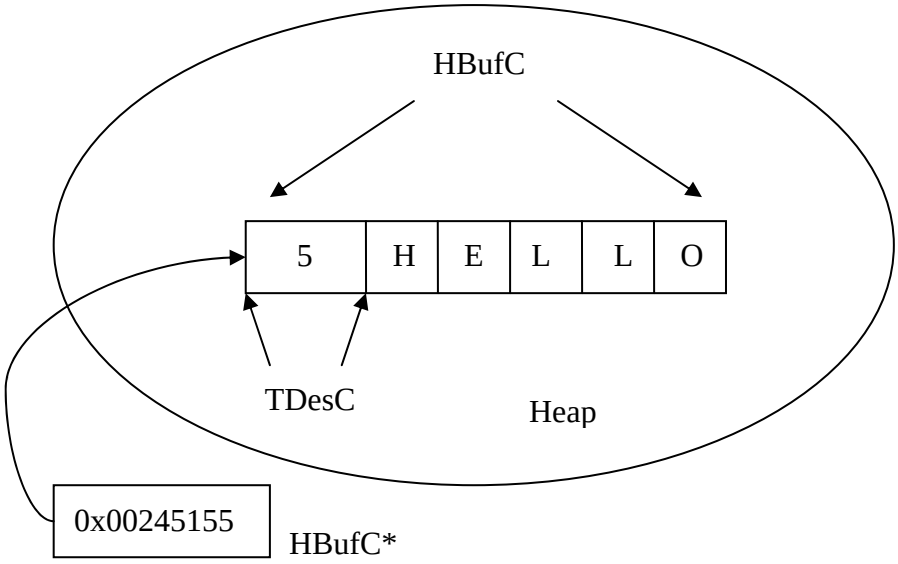


Figure 2.3: Representation of a Heap Descriptor

3. Pointer Descriptor

Data is separated from the descriptor object. The descriptor object lives in program stack while the data may lives in heap or program stack.

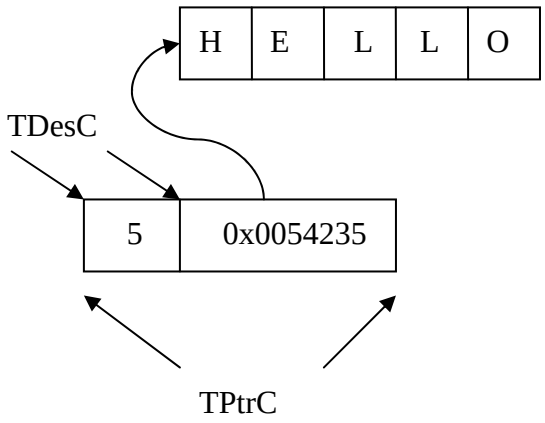


Figure 2.4: Representation of a Pointer Descriptor

2.5.1.3 The Hierarchy of Descriptors

Figure 3.4 shows the hierarchy of descriptors. All the descriptors, modifiable, non-modifiable, buffer-based, heap-based and pointer-based, are all derived from the abstract base class TDesC. As implied by the name, TDesC is non-modifiable. Other modifiable descriptor classes have some extra information and functions to make them modifiable. The abstract class facilitates the design of function and parameter passing.

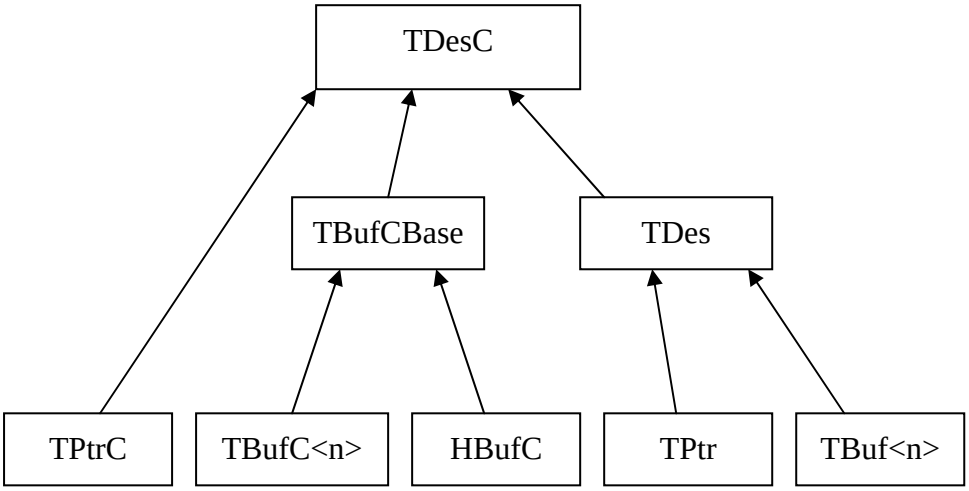


Figure 2.5: The Hierarchy of Descriptors

2.5.2 Cleanup Stack

Memory is of very limited size in most mobile device. It is even more treasurable in mobile phones. If we allocate memory and do not free them after finished using them, we may run into out-of-memory problem. Unlike desktop machines, which could be readily restart when there is out-of-memory problem caused by extensively not freeing memory after use, mobile devices are expected to work for a long period of time before they are switched off or restarted. Therefore, freeing unused memory becomes an important issue. If memory is continuously being allocated without freeing up, the mobile device is under the risk of out-of-memory problem.

In C++, a **delete** function is provided so as to free up heap memory that is allocated by objects. It seems nice with the function to free up unwanted memory, but there is still the chance that **delete** cannot free those unwanted memory. Consider the following code fragment:

```
CX* x = new (ELeave) CX;
x->UseL();
delete x;
```

Code fragment 2.1: Possible cause of Heap Failure

where CX is a class, x is an automatic variable allocated from heap by the identifier **new** and UseL() is a function that **leave** (terminates upon encountering unexpected error). In the above implementation, if UseL() does leave when it is executed, the automatic variable x will never be able to be freed since the pointer pointing to that memory is destroyed. This will cause a heap failure (heap memory cannot be freed) and may result in out-of-memory problem in a long run. This introduces a cleaning mechanism using the Cleanup stack.

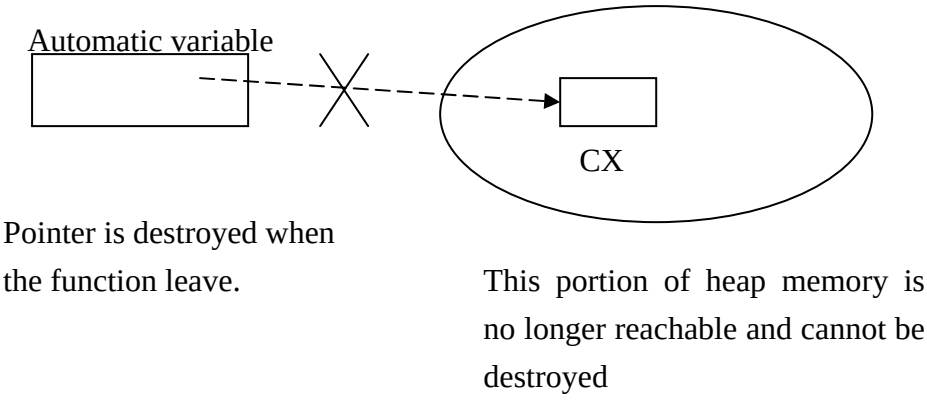


Figure 2.6: Heap Failure caused by a Leaving Function

The problem of heap failure is caused by automatic variables. Cleanup stack pinpoint this problem by providing a place for holding these automatic variables. With the existence of Cleanup stack, code fragment 3.1 can be rewritten as follow:

```
CX* x = new (ELeave) CX;
CleanupStack::PushL(x);
x->UseL();
CleanupStack::PopAndDestroy();
```

Code fragment 2.2: The use of Cleanup Stack

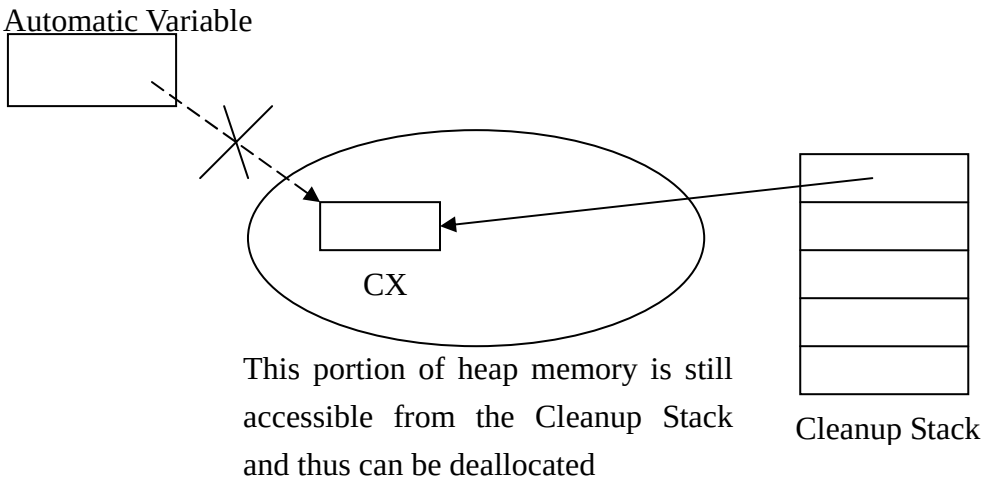


Figure 2.7: How Cleanup Stack help to ease Heap Failure

Cleanup stack will free up all the heap memory that are addressed by the pointer stored in the Cleanup stack. Thus, after we created an automatic variable, we push it onto the Cleanup stack. If the program leaves, then the allocated heap memory will be freed by the Cleanup stack. Therefore, using Cleanup stack can ease the problem of heap failure.

2.5.3 Two-phase Construction

The Cleanup Stack provides a safe mechanism to prevent memory leak. However, there is still some situation that memory may leak.

```
CY::CY()     // Constructor of CY
{
      iX = new (ELeave) CX;
}
```

Code fragment 2.3: Construction that may leave

When an instance of CY is being instantiated, there may be some problem about memory leak:

1. If the allocation of CY fails, everything leaves and there is no problem

2. If the allocation of CX fails, the codes leaves, but CY is not yet pushed onto the Cleanup Stack, the memory allocated to CY leaks.

3. After the initialization of CY and CX, any call to function that leave causes no problem, since CX and CY are on already on the Cleanup Stack.

The problem occurs when a leaving function, including the construction of another object, is executed. Immediately after CY is instantiated, there is not yet a chance to push it onto the Cleanup Stack. If a leaving function is called here and it does leave, CY will not be able to be de-allocated. Therefore, Symbian OS introduce another rule to prevent memory leaks. The rule is called the Two-phase Construction. This rule, together with the use of Cleanup Stack, fully prevent memory leak from occurring in Symbian OS.

Two-phase construction states that all constructors must not leave. This means that no leaving function, including the construction of other object, can be performed in the first phase of the construction process. All leaving function can only be called in the second phase of the construction process.

Two-phase construction divides the construction process into two phases. The first phase is the normal construction, denoted by the object's name, for example CY::CY(). The second phase is usually denoted by CY::ConstructL(). In fact, it is a leaving function. All the operations that might leave should only be done here.

The construction process with two-phase construction rule is like follow:

1. Create the object, for example CY, using the first phase constructor
2. Push the object onto the Cleanup Stack
3. Call the second phase constructor to perform operation that leaves

In this way, if any of the operation in the second phase of the construction process leaves, CY will still be able to be deallocated since it is already on the Cleanup Stack.

2.5.4 Active Object

Symbian OS is a multi-tasking operating system. However, unlike most operating system, it does not use multi-threading to achieve the purpose of multi-tasking. Instead, it utilizes a mechanism called Active object to perform multi-tasking.

With a view to the system overhead of multi-threading, Symbian OS turned to another approach that requires lesser system overhead. In using active object to perform multi-tasking, there is only one thread. In the thread there is an active scheduler together with one or more active objects. The active objects send out requests. These requests will then be scheduled by the active scheduler so that these requests can be served in an interleaved manner.

The active scheduler schedules to serve the requests sent out from active object in an non-preemptive manner. Therefore there is no mutual exclusion code needed in using active objects. This further reduces the code complexity while achieving multi-tasking.

2.6 Summer Work on Symbian OS

In the summer, we learnt to program for the Symbian OS. We wrote two little applications to demonstrate what we have learnt and the computing power of this OS.

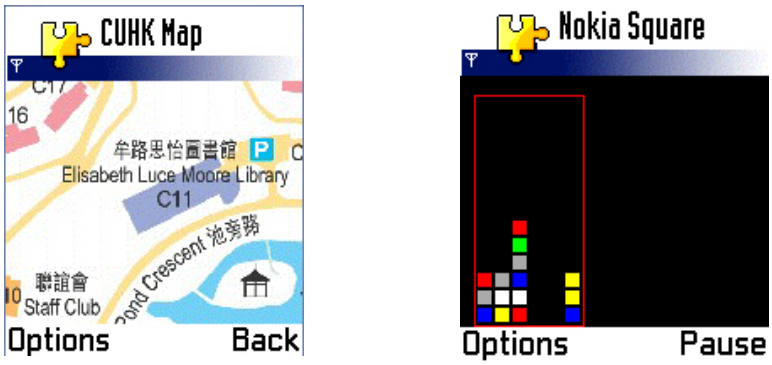


Figure 2.8: Programs written for the Symbian OS

The CUHK Map shown in figure 2.8 is a program that displays the map of the CU campus. User can scroll in the four directions to view the map. Nokia Square on the right hand side of figure 2.8 is a game that looks like Tertis. The one shown in figure 2.9 is a program written to show the various GUI components that can be used in Symbian OS.

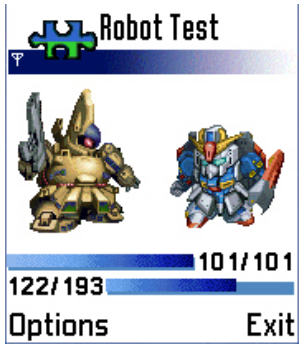


Figure 2.9: A program for illustrating the power of GUI components of Symbian OS

Having introduced the operating system we are going to use, we will next describe the basic GSM knowledge we will use in our project.

3. The GSM Technologies

3.1 Cellular Technology

GSM networks are hierarchically structured. In a GSM network, there is at least one administrative region, which is the Mobile Switching Centre (MSC). Each administrative region consists of at least one Location Area (LA). Each location area in turn is made up of several cell groups. Each of these cell groups is assigned to a Base Station Controller (BSC). Figure 3.1 shows the system hierarchy of the GSM network

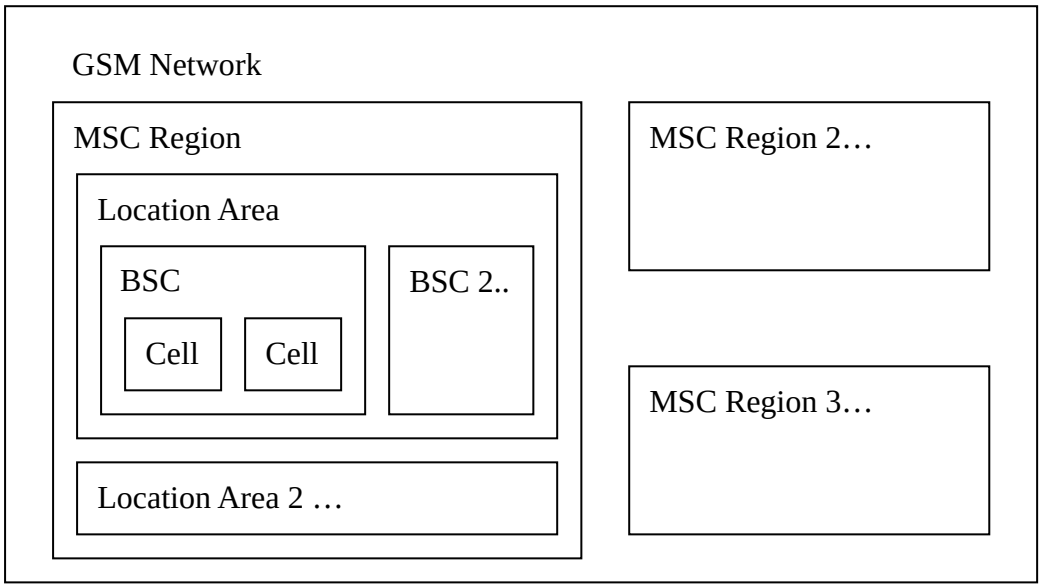


Figure 3.1: System hierarchy of the GSM network

Because of the limited frequency bands, GSM system only allocated 25MHz in the 900MHz frequency range. Maximum of 125 frequency channels each with a carrier bandwidth of 200kHz. Therefore GSM service providers must reuse their assigned frequency repeatedly. This give rise to the formation of cell clusters. For example, if a GSM operator is assigned k number of different frequencies that it can use to provide service, it has to reuse the frequencies. In order to prevent inter-channel cross-talk, cells using the same frequency band should be placed as far apart as possible. Figure 3.2 shows how frequencies can be reused in three cases, in each case the GSM operator is assigned different number of frequencies that it can use (indicated by k). Different

cell clusters are formed in each case.

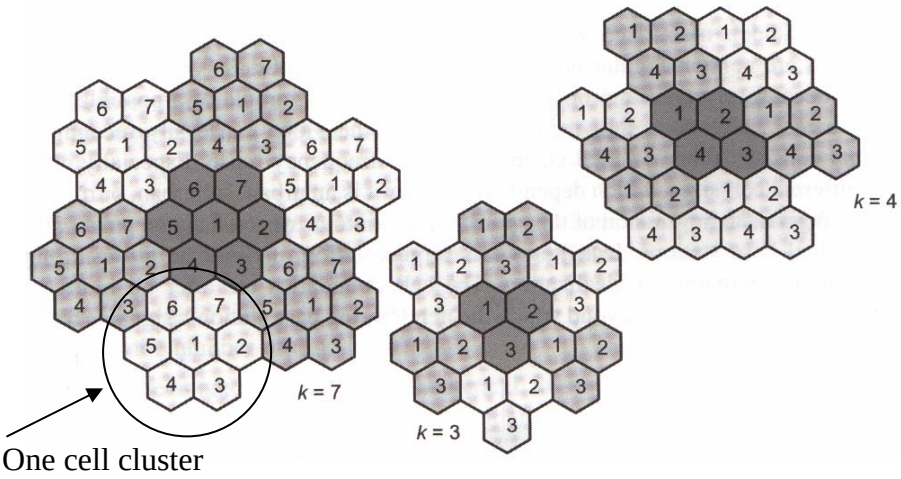


Figure 3.2: Frequency reuse and cell cluster formation

3.2 System Architecture and Addressing

As we mentioned in the previous subsection, GSM system is structured hierarchically. Figure 3.3 shows the physical structure of the GSM system architecture with essential components. Some important components are described in the previous subsection, while others are not very important to our project so we just ignored them.

Since the system is in a form of hierarchy, there should be some way to identify each component among the whole system. There are some international standard in addressing each mobile station, for example the International Mobile Station Equipment Identity (IMEI), which utilizes the Type Approval Code, Final Assembly Code and Serial Number to uniquely identify a mobile station internationally. However, in this project we only concentrate on how to identify each mobile station locally, but not internationally, so we do not need the IMEI, which is too detailed to be used. Instead, we concentrated at the Location Area Identifier (LAI) and the Cell Identifier (CI), which when combines together, can identify a local mobile station uniquely.

LAI consists of two portions, the Country Code (CC) and the Mobile Network Code (MNC). These two codes together forms an identifier which of maximally a 5 decimal digits number. For the CI, it is of

maximum length of 16 bits in size.

LAI uniquely identify a location area in a GSM network while the CI uniquely identifies a cell in a base station controller's domain.

3.3 Cell Selection and Reselection

A mobile phone must periodically measure the strength of the signals emitted from the base stations nearby. Base on the measurement a mobile phone select a cell which gives the best reception. After connected to a cell, accessing a service become possible. However, when the mobile phone moves along a path, the strength of signal received from the base station that it is connected to may drops, which may due to several reasons like it is moving away from that base station or there is some environmental factor that distort the reception. In this case, a mobile phone must reselect another cell which gives a better strength of signal. This process is called the cell reselection. Cell reselection is very common, when we travel around some place we usually experience quite a number of cell reselection event, since cell are of limited size. When we reach the edge of a cell, a mobile phone will reselect another cell that gives a better reception in order to guarantee the quality of service provided to a user. When a cell reselection event occurs, the mobile phone will transit from one cell to another cell, thus receiving signal from another cell which is of different LAI and CI.

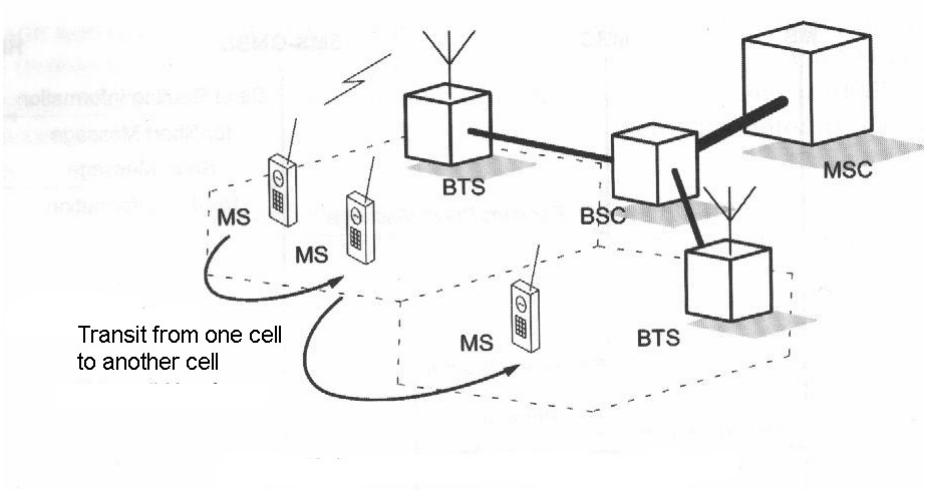


Figure 3.3: Cell reselection when traveling from one cell to another cell

4. Currently Existing LBS Solutions

There are currently two major streams of LBS solution available. They are the Global Positioning System (GPS) and SMS Query using specially designed SIM card. GPS is already very mature and is widely used in industrial and commercial fields, while the latter one is still very new and is still under the development process. In the following subsection we will give a brief introduction on the two methods and compare their advantages and disadvantages.

4.1 The Global Positioning System

The Global Positioning System is a satellite-based navigation system made up of a network of 24 satellites. The satellites orbit around the earth in 12 hours. There are totally 6 orbital planes which are equally spaced (separated at about 60 degree from each other).

Figure 4.1: 24 satellites and 6 orbital planes for GPS system

21 GPS satellites and three spare satellites are in orbit at 10,600 miles above the Earth. They are set to be in those positions so that there will be four satellites above the horizon at any particular location on the Earth. Each satellite continuously broadcast a signal representing its position. Then GPS receivers on the Earth receive any three of the four satellites that is above the horizon and calculate it position by using triangulation. The result is then represented in form of geographic

location, which includes longitude and latitude. The signal from the fourth satellites can be used to find out the attitude.

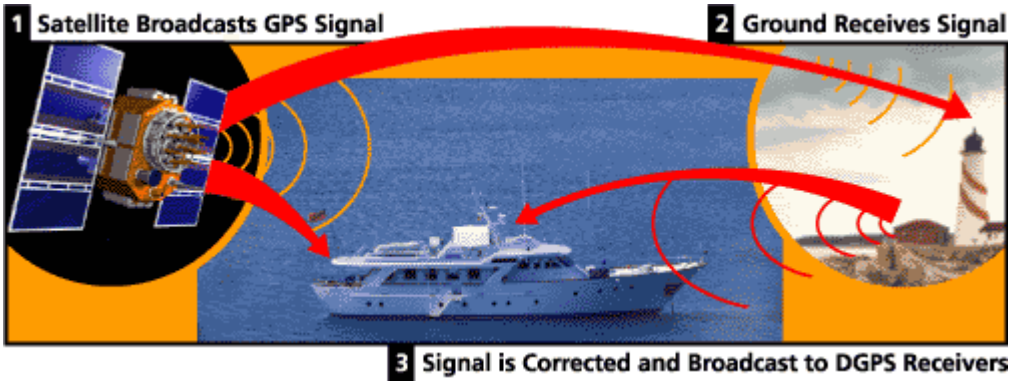


Figure 4.2: How GPS works

GPS is widely used in science, in sea navigation and in air (flight) to determine the current location and the route to take to destination. Furthermore, GPS is used militarily. There are still many applications but mostly not targeted to the general citizen.

Figure 4.3: PDA equipped with GPS receiver for location determination

4.2 SMS Query using Specially Designed SIM Card

This system is much simpler than the GPS system. It involves no satellite. What it needs are specially designed SIM card and some GSM base stations. It works as follow: The application drive the mobile phone to constantly send out SMS query message. Then the nearest base

station will reply the query with a location string and some information of the current location of the user. Since the GSM base stations are deployed in a systematic manner, each base station can represent a specific region of area. So, when a base station replies an SMS query, it is sure that the user is inside that region, otherwise the SMS query would be replied by another base station which is nearer to him. A specially designed SIM card is needed so as to accomplish this task. This specially designed SIM card is used to send out special SMS query to GSM base station. This LBS solution uses simple equipment and would be more easily accepted by the general public.

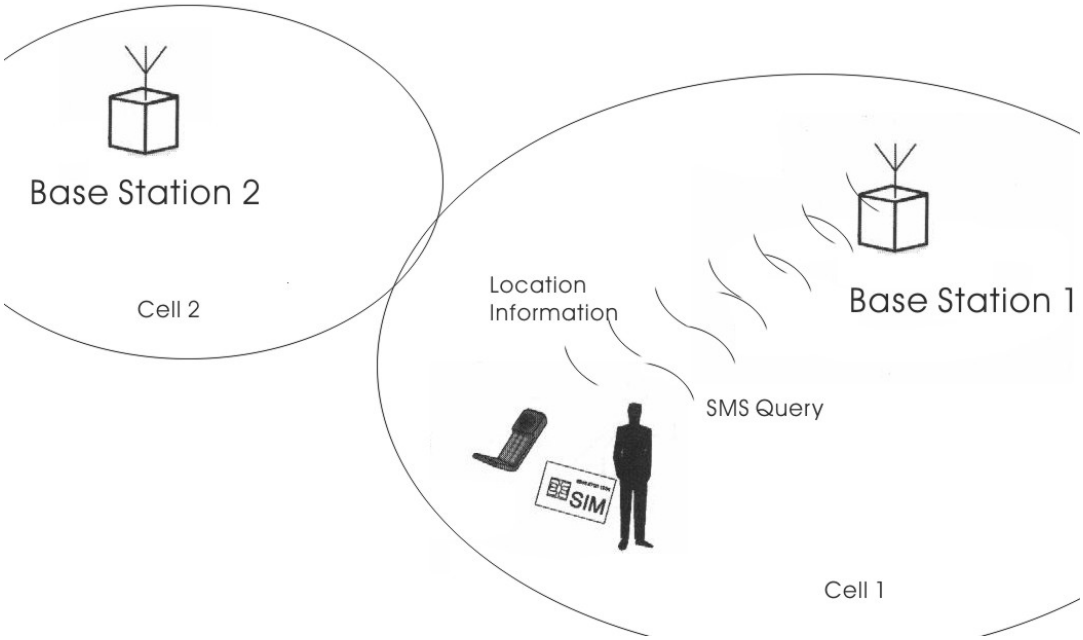


Figure 4.4: The operation of SMS Query for location information. The mobile phone actively sends out queries

4.3 Comparison of the Two Methods

The above methods mentioned are nice. They have their relative advantages and disadvantages.

The accuracy of GPS is generally high. It is about 10 to 100 metres with most equipment. This accuracy can be improved to 1 metre with military-approved equipment. For the second method, the accuracy greatly depends on the cell size and the signal strength of the base

station. If the cell size is small, the accuracy can be high (within 50 metres). However if the cell size is large, the accuracy can be reduced to about 500 metres.

Both methods require extra hardware support. GPS users have to get a GPS receiver installed while the latter method requires user to have a specially designed SIM card.

The draw back of using GPS is that GPS signal would be seriously distorted by severe weather condition. Since signal has to be sent from space to the Earth, a severe weather condition, for example thunderstorm, would distort the signal, making the location-determination process inaccurate. Secondly, GPS signal is not reachable in closed environment, like inside a building. Furthermore, three satellites must be visible by the GPS receiver in order to provide the service. It is not very probable in densely populated urban area.

The SMS Query method does not have this constraint, since GSM signal can reach most part of the building, users can still query their current location. However, the major draw back is that the active role of the mobile phone requires it to constantly send out SMS query, which will occupy the communication channel of the mobile phone, other communication, for example making a phone call, will not be available then.

5. Location-based Service using LAI and CI

The two methods mentioned in the previous section are nice, but they require special hardware device like GPS receiver and specially designed SIM card. In order to provide location-based service with the minimal use of special hardware and overhead, we turned to an approach that tries to retrieve the Location Area Identifier (LAI) and Cell Identifier (CI) from the base station our mobile phone is connected to.

5.1 The Principle

In order to provide service with good quality, mobile phone service providers deploy GSM base station in a way that most of the area people may go to will be covered by one or more than one GSM cell. Each GSM cell covers a region of area which may overlap with other cells. Each mobile phone, when being switched on, will try to connect to one and only one GSM base station in order to access their service. Mobile phones would connect to a base station that gives the strongest signal strength to the location where the mobile phone is at. In most of

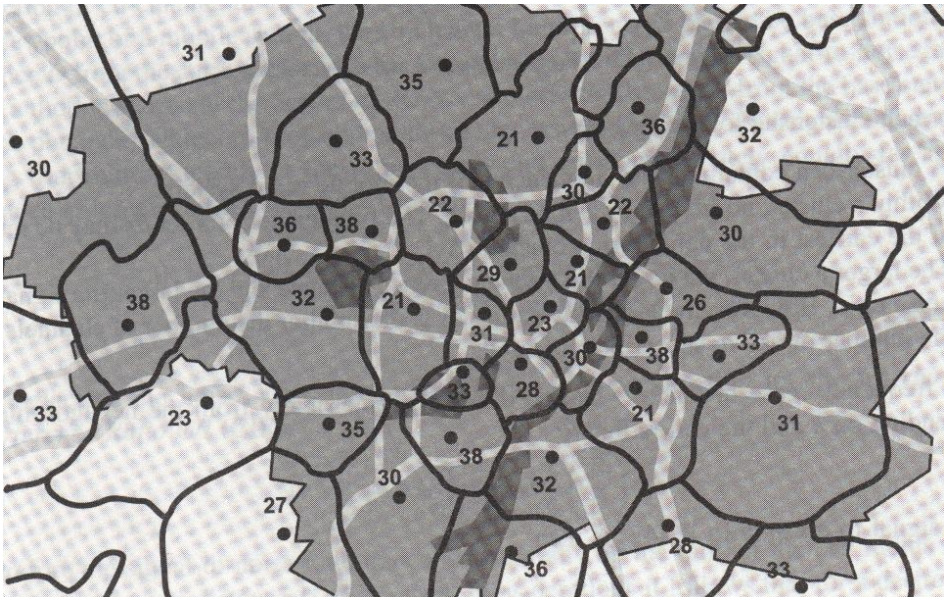


Figure 5.1: Area covered by cells. The dots inside cells represent the base stations

the case, the nearest base station would give the strongest signal strength to the mobile phone. Therefore, retrieving the LAI and CI of the base station that a device is currently connected to can tell us which cell region we are currently in. This in turn gives us an approximation of the current location of the mobile phone.

A GSM base station continuously sends out data and control frames. In the header of each frame, there are fields storing the information of the base station which send out the frame. One of the information is the base station identification which includes the location area identifier (LAI) and cell identification (CI) of that station. These information are transparent to mobile phone users. Using Symbian OS, a mobile phone can retrieve these data for special use. We try to use this approach to determine our current location.

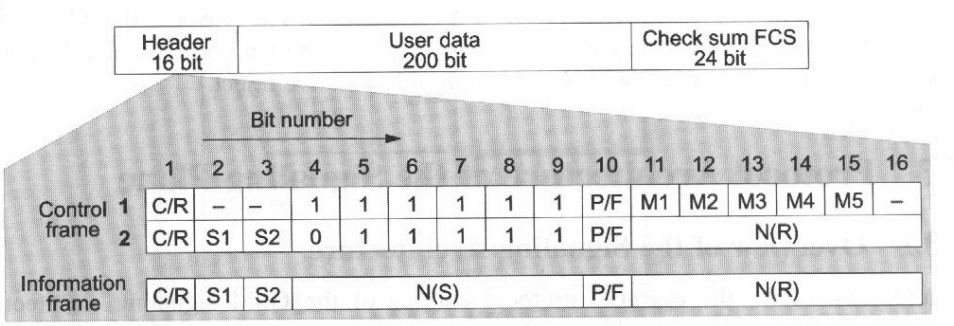


Figure 5.2: Structure of a GSM frame under RLP protocol. Header includes base station information

We try to discover the LAI and CI from the signal received by the mobile phone. Since a mobile phone can at most connect to one GSM base station which gives the strongest signal (which usually implies the nearest GSM base station) by retrieving the location area identifier and cell identifier pair of that GSM base station, we could approximately determine our current location. In the following sections, we will explain our main principle and what have to be done in order to provide this location-based service.

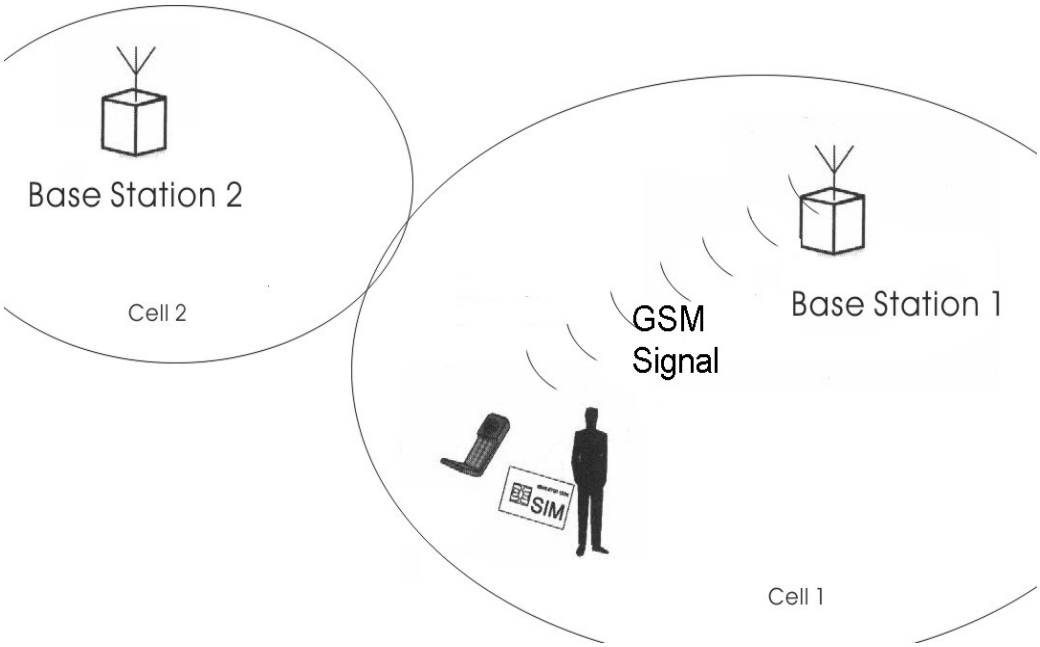


Figure 5.3: Retrieve LAI and CI passively from GSM broadcasting signal.

5.2 Development of the Data Collection Kit – GSM Status

To retrieve the currently LAI and CI from the GSM signal, we need to write a program which runs on Symbian OS and perform the required task.

We used Nokia Series 60 platform, which is in fact Symbian OS version 6.1 with a graphical user interface specially designed by Nokia, to perform our task. Inside the libraries provided by the system development kit, there is one function which can perform the task of retrieving the cell identification numbers. However, owing to some special reason, Nokia removed the header of this function from the system development kit, making it inaccessible by developers. In order to use this function, we have to re-activate this function by adding the header back to the program. In the older Communicator 9200 SDK, the function was not yet hidden from developer's access. Therefore, by copying the header files etel.h and etelbgsm.h from this SDK to our SDK and link them to our program, we can re-activate the function for our project to use.

With this function, we wrote a program that retrieve the current LAI and CI pair.

Figure 5.4: GSM Status that retrieve current cell information

The program retrieves the current cell information on every one-second interval. It displays the data onto the screen. Then it compares the pervious data with the newly-retrieved data, if a cell change occurs, it pops up a dialog box telling the user that a cell change has occurred. From figure 5.4, we can see that three information are retrieved. They are the LAI (displayed as “Location”), CI (displayed as “Cell ID”) and relative signal strength (displayed in them of percentage). Whenever any of the information changes, the program updates the data in the one-second interval’s time.

5.3 The Data Collection Methods

In order to investigate whether the use of LAI and CI is appropriate to serve as the foundation of a new direction of LBS, we have to look into the cell distribution, signal strength and the accuracy over an area.

To figure out the cell distribution within an area, we have to use our data collection kit, the GSM Status, together with some data collection rule, to achieve our goal.

GSM Status returns us the LAI, CI and relative signal strength at the particular point where the mobile phone is at. However, as we all know,

there is overlapping of cells over the whole region of area, a data retrieved at that particular point in the area may only represent part of the actual data that we need to collect. Therefore, to figure out the cell distribution and boundaries within the whole area, we need some rule in collecting the data.

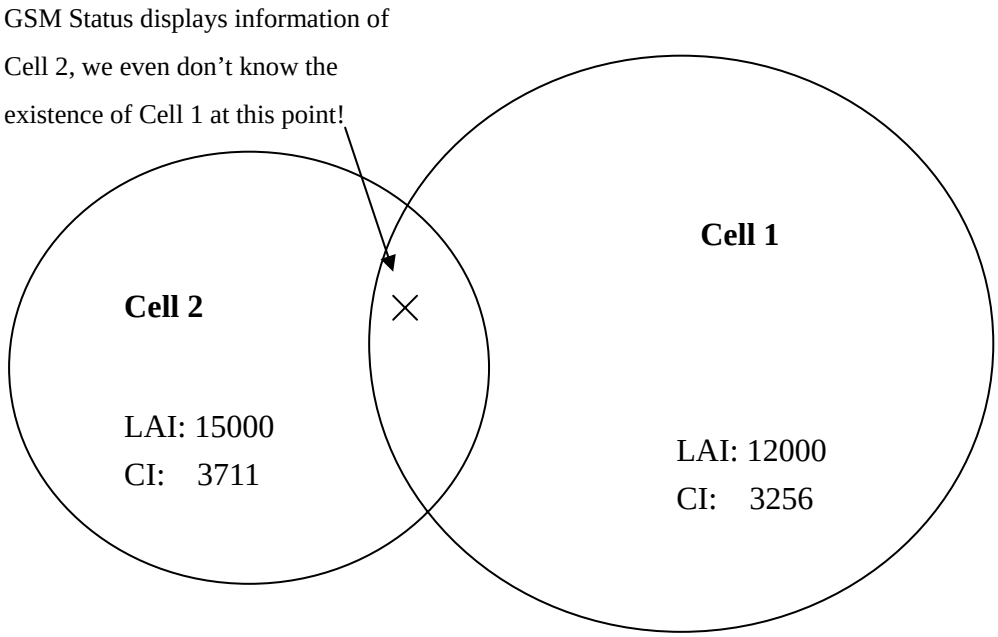


Figure 5.5: We need a rule to collect data so as to figure out all the hidden data

We proposed two rules to collect cell information. They are the static method and the cell change method. The two methods are explained in the following subsections.

5.3.1 The Static Method

The static method is a rule for use to collect cell information using GSM Status. In this method we first define a set of important points on the map of the area that we are interest. Then we use GSM Status to collect data at those set of points. At those points, we wait for a sufficiently long period of time to observe the cell information. If the point is at the overlapping region of two or more cells, we may experience cell change event during this period of time since the strength of the signal emitted by base station is dependent on the environment, for example the existence of an moving object like car. Any environmental change would have some effect of the signal strength. If the signal strength of the base station we are currently connected to drops below the strength of signal

emitted by another near by station, the mobile phone will connect to that station instead, to guarantee communication quality. Therefore we may be able to determine whether we are at the overlapping region and can discover the existence of both cells.

By defining points dense enough and observing the data for a longer time, we can accurately determine the distribution of all cells within a region.

In fact, we can make the result even more accurate. We can repeat the experiment several times at those selected points and find out the probability of getting particular cell information. Then we may draw a probability contour line map to figure out the region covered by a particular cell with different probability. However, since this method requires a very long experimental time, we did not perform this (plotting probability contour line map) in our experiment.

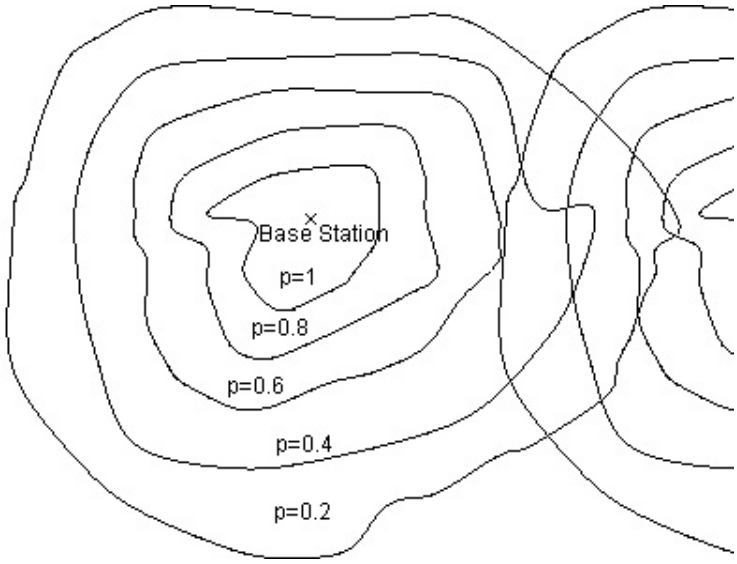


Figure 5.6: Probability contour line map with probability p that we are in a particular cell

5.3.2 The Cell Change Method

Another method we propose is the cell change method. In this method, we do not define any set of points to be taken as experimental points. Instead, we walk around the whole area to detect cell change events. Since our data collection kit is capable of detecting cell change event, we can use the cell change event to figure out the boundaries of the cells inside the area to be investigated. By walking along all the main path in the area, we may find out most of the place where cell change take place. By repeat the experiment several times we may improve the accuracy of the experiment and can figure out the boundaries of cells.

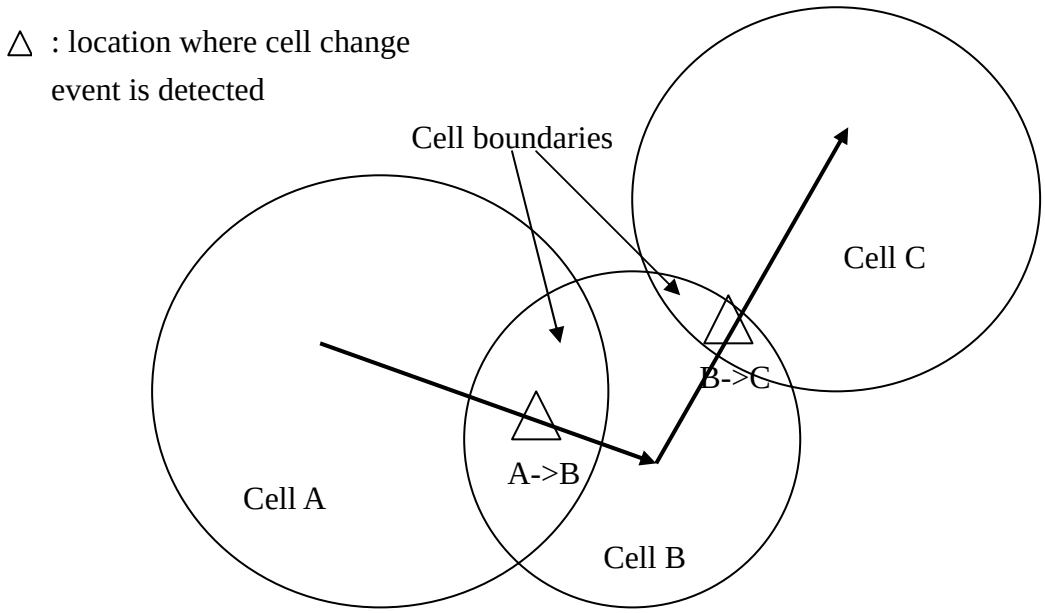


Figure 5.7: By walking along the main path in the area, most cell boundaries can be detected

5.3.3 The Comparison of the Two Methods

The two data collection rules have their advantages and disadvantages. We summarized them in the following table

:

	Static Method	Cell Change Method
Advantages	Results very accurate at those selected points	Can figure out the boundaries of different cells in the area
	Experiments only done on those selected points	Fast, no need to wait for a long time to get the result
Disadvantages	Takes a longer time	Boundaries detected are regions instead of sharp lines
	Cannot figure out the distribution of cells clearly without dense selected points	Have to walk through the whole area several times

Table 5.1: Comparison of the two data collection rules

There are relative advantages in each method we mentioned, in order to completely find out the cell distribution in an area, we used both the methods in our project.

6. Location-based Service in 2-dimensional Space

6.1 Motivation

Most of the current LBS applications are targeted in the 2-dimensional space. We would also like to work on the 2-dimensional field, which dominates the market of LBS application.

Hong Kong is an international city. Many tourists from all over the world enjoy traveling in Hong Kong. With technology advancement, it is supposed that we should have an electronic tour agent to guide tourists while they are traveling around Hong Kong. However, tourists may not have a GPS receiver to utilize the existing LBS. Therefore, we would like to develop an electronic tour agent running on mobile phones that utilize the Symbian OS. Therefore, we first choose the campus of The Chinese University of Hong Kong as a testing site. We would like to know whether the application of LAI and CI would be capable in helping us to build our system.

6.2 The Experiment

In order to investigate the capability of using LAI and CI for determining the current location, we have to figure out the cell distribution, cell size, degree of overlapping in the campus first. Then, we will plot a cell-to-location map for future use in building our system.

In this experiment, we investigated the cell distribution of two local telcos. They are SmarTone and Peoples. We used different approach mentioned in the previous section in different telcos. For SmarTone, we used the static method in collecting the cell information; for Peoples, we turned to the cell changed method. We repeated the experiment five times in order to obtain a better accuracy.

6.3 Expectation of the Experiment

Before we do our experiment, we have the following expectations:

- Cells are of similar size
- The portion of cell overlapping is small and only occur at the edge of the boundary
- No large cell completely covering a smaller cell

- Cells can be modeled as hexagonal shape cluster covering the whole area.

6.4 Results and Statistic

Distribution of cell identifiers in the CU campus for SmarTone:
(LAI is the same throughout the CU campus, number represents CI)

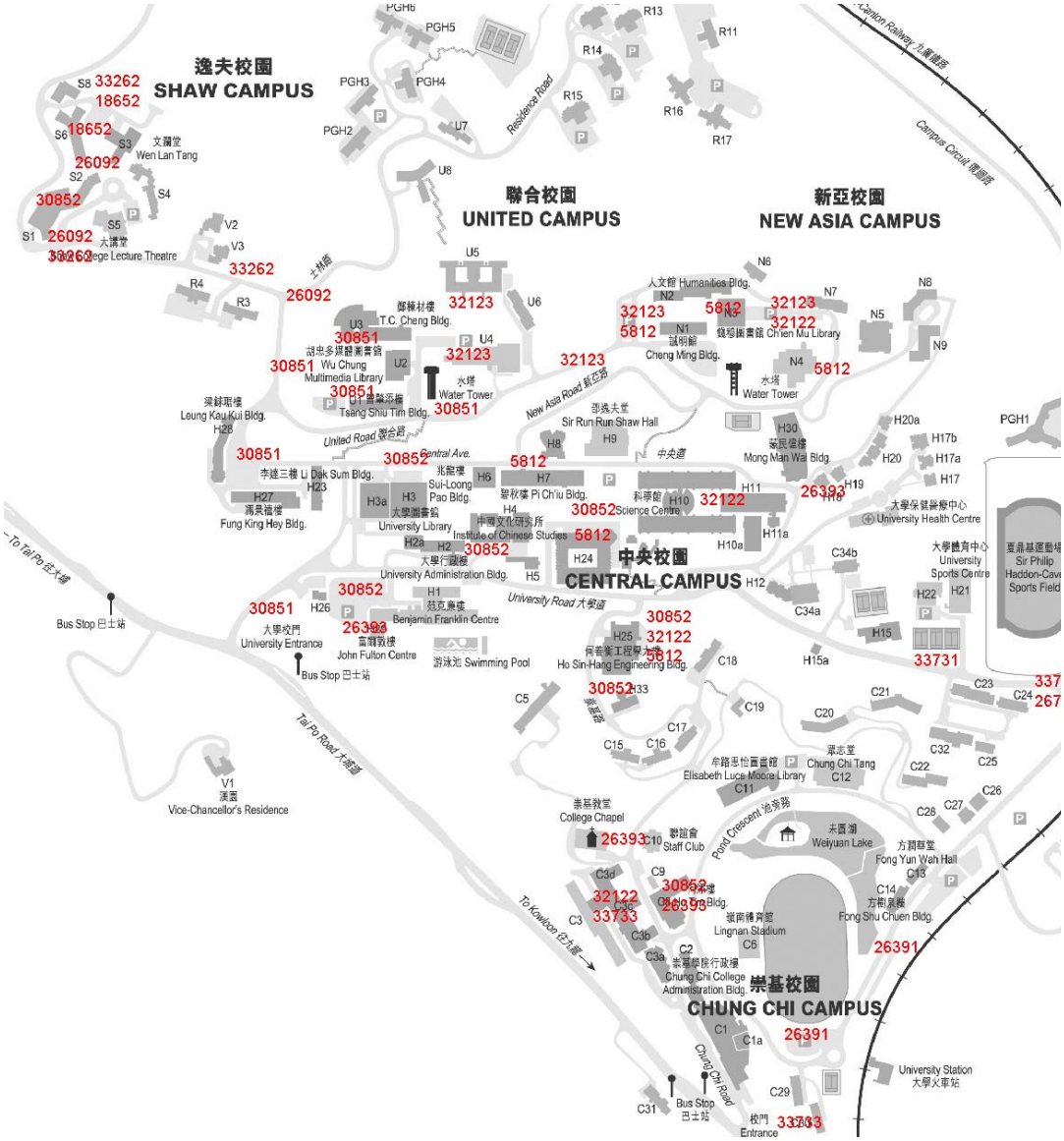


Figure 6.1: Experimental results for SmarTone

Outline of cell boundaries for SmarTone in the campus.
(The number represents the cell identifier)

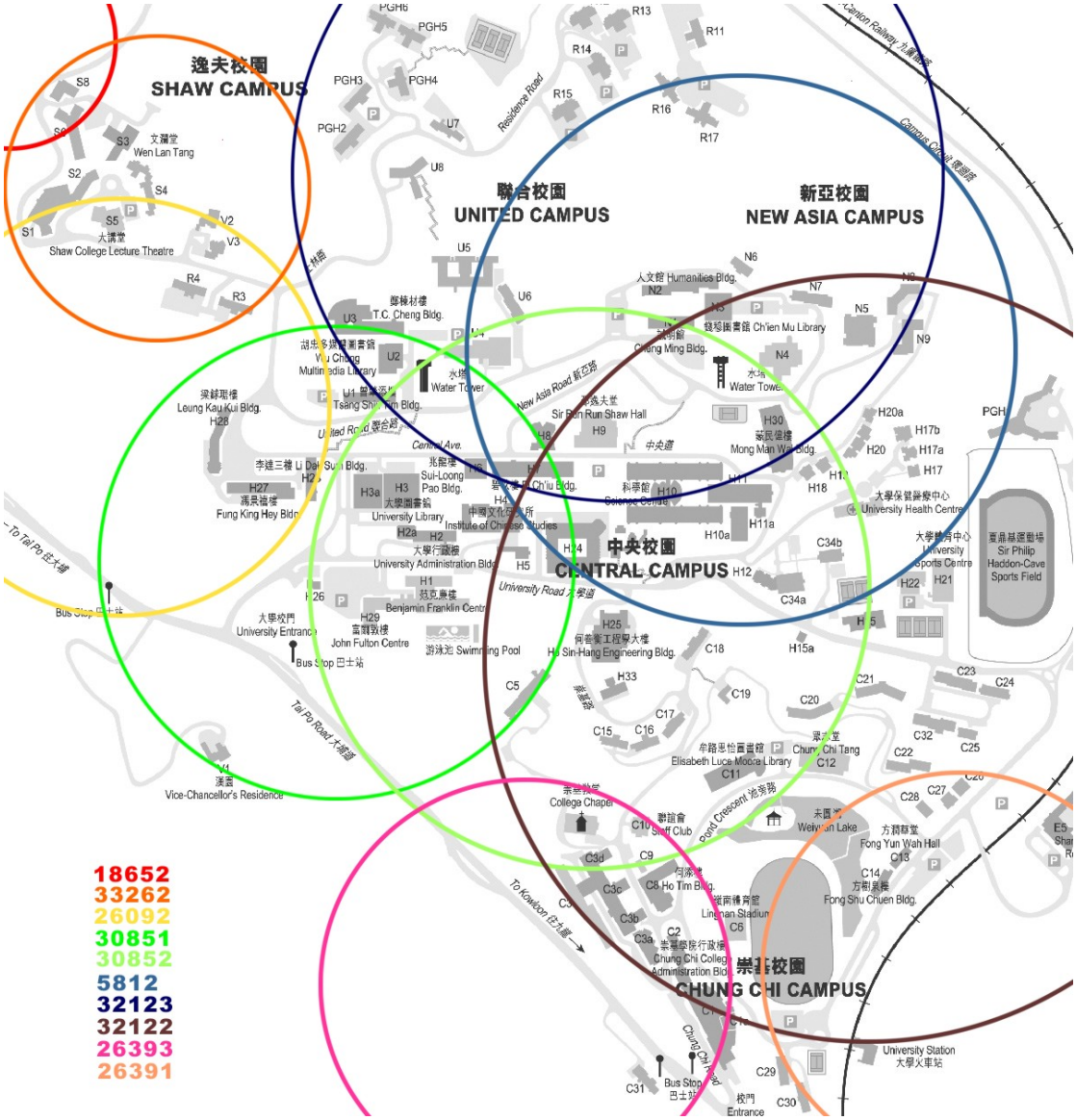


Figure 6.2: Approximated cells distribution in the CU campus for SmarTone

Distribution of cell identifiers in the CU campus for Peoples:
(LAI is the same throughout the CU campus, numbers represents CI)

Figure 6.3: Experiment result for Peoples

Outline of cell boundaries for SmarTone in the campus.
(The number represents the cell identifier)

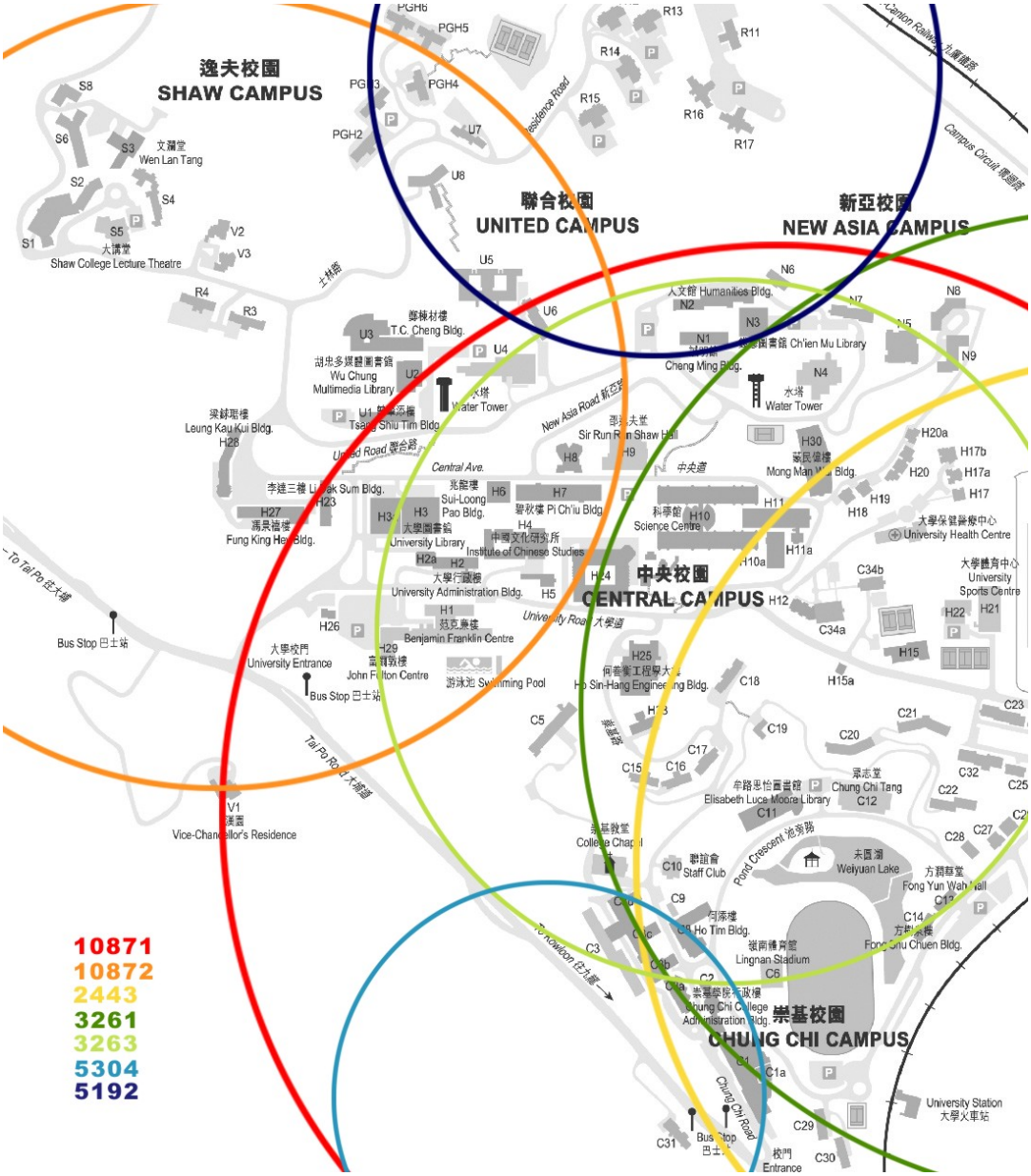


Figure 6.4: Approximated cells distribution in the CU campus for Peoples

6.5 Conclusion of the Experiment

After performing the experiment, we found there are some discrepancies with our expectations.

Firstly, cell varies greatly in size. For example, from the result of the experiment done on Peoples (figure 6.4), the cell in blue colour is very small compared with the orange one.

Secondly, the scale of cell overlap is quite large. From the result of SmarTone (figure 6.2), the light blue cell overlaps greatly with the light green one.

Furthermore, there are large macro-cell completely encapsulating a smaller micro-cell. From figure 6.4, a light green cell is totally within the red cell.

Also, some cell covers an area that is too large to be used in accurately determine the current location. For example, in figure 6.2, a brown cell covers a quite large area (including the Chung Chi campus and part of the main campus). This cell, without other cells' assistance, cannot give enough information for use to accurately determine our current location. An example is that: when we are at the CU bus station near the KCR station, we may receive the cell information of the brown cell (CI = 32122). However, the brown cell covers a very large area. With only this information we cannot say for sure that whether we are at the CU bus station or at the University sport centre.

Lastly, cell may change shape under different environmental condition. This may make the cell boundaries slightly different from the expected ones.

To conclude, there are several potential difficulties in using the LAI and CI to determine accurately the exact location of a mobile user. It is because the cell size is controlled by the telco, and those cells are usually too large to be used accurately in this way. Furthermore, mobile phones currently can only connect to one single mobile station. No other information can be retrieved other than the information of the base

station we our currently connected to. All these make the use of LAI and CI not accurate to be used in the 2-dimensional space. Because of this inaccuracy, we turned to the 1-dimensional space LBS.

7. Location-based Service in 1-dimensional Space

7.1 The Principle

There are quite a number of technical problems we have to solve in the 2-dimensional space. Without the help of telcos it seems not very possible to provide LBS that are accurate enough to be used by the general public. Therefore we turned to the 1-dimensional space.

1-dimensional space here we mean a line. Since cell may vary greatly in size, the information of exactly which point we are currently at in a cell seems not very useful. We are more interested in the event of cell change in the 1-dimensional space.

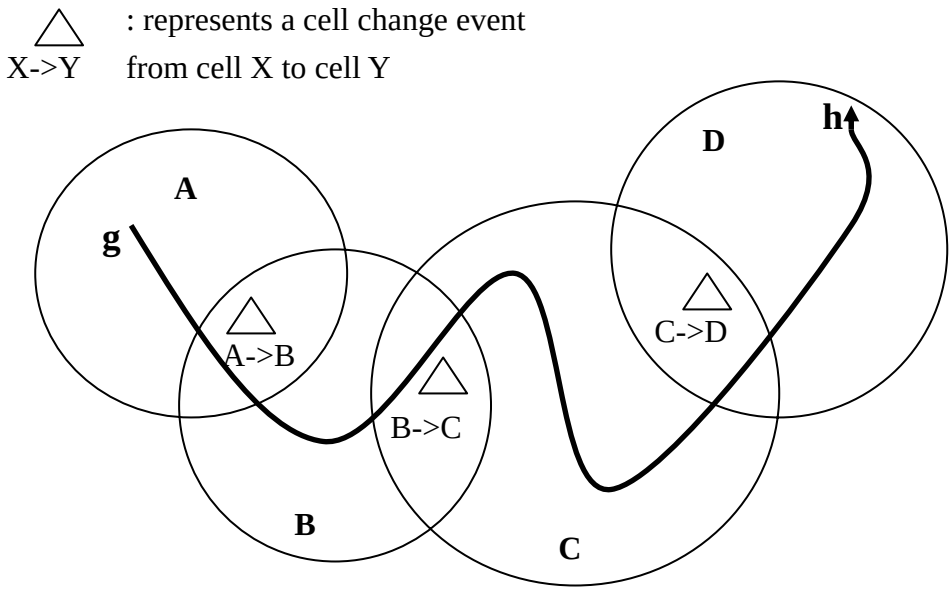


Figure 7.1: Cell change events happened in a 1D space

In the one dimensional space, we are interested in the “entrance” of a new cell. Other information becomes not important to us. In other words, we only want know when we transit from one cell to another cell.

In a 1-dimensional space, for a particular route, when it goes from one place to another place, it must pass through a set of cells that are located along the path. For example, in figure 7.1, a route from **g** to **h** passes through four cells: A, B, C and D. No matter how many times we go from **g** to **h**, if we take the same route, we must pass through these four cells. Thus the three events (A->B, B->C and C->D) must occur during the journey from **g** to **h**. With a route and a set of cell change events, we can say for sure that which region we are currently in with a very high confidence For example, when we detect a cell change event of P->Q, then we know that we are currently in the realm of base station Q, and we were in the realm of base station P..

Therefore, a pre-defined route can eliminate all the uncertainties and inaccuracies we have had in the 2-dimensional space.

With the principle of LBS in the 1-dimensional space, we turn to something along a path - rails and buses. Travelers, when touring around Hong Kong, must take public transports. However there are too little information can be retrieved from the public transport system. For example when a tourist want to go to Clear Water Bay, he takes an MTR, where should he change for other public transport, say, minibus? Another case is that when a tourist is traveling on an MTR, he is passing through Causeway Bay station, how can he know more about the sightseeing site there? Any shopping mall or restaurant there? Where and when to change train? With a view to this, we proposed to develop an LBS system for the rails (MTR and KCR) to give more information about the current location where they are in, and to prompt them to change for other trains. We will still use the LAI + CI information in Symbian OS to achieve this.

7.2 The Experiment

This time we do our experiment on two rail systems – MTR and KCR. We used our data collection kit to note down any cell change event happen during the journey. We travel to and fro different stations for more than five times so as to improve and verify the results.

To illustrate the experiment more clearly, refer to figure 7.2. In figure 7.2, we travel from station A to station B. Since there are three cells in

between the two stations, we should experience three cell change events during the journey from A to B. We would then note down these cell change events between stations. These data would then be used to develop our system for the rail.

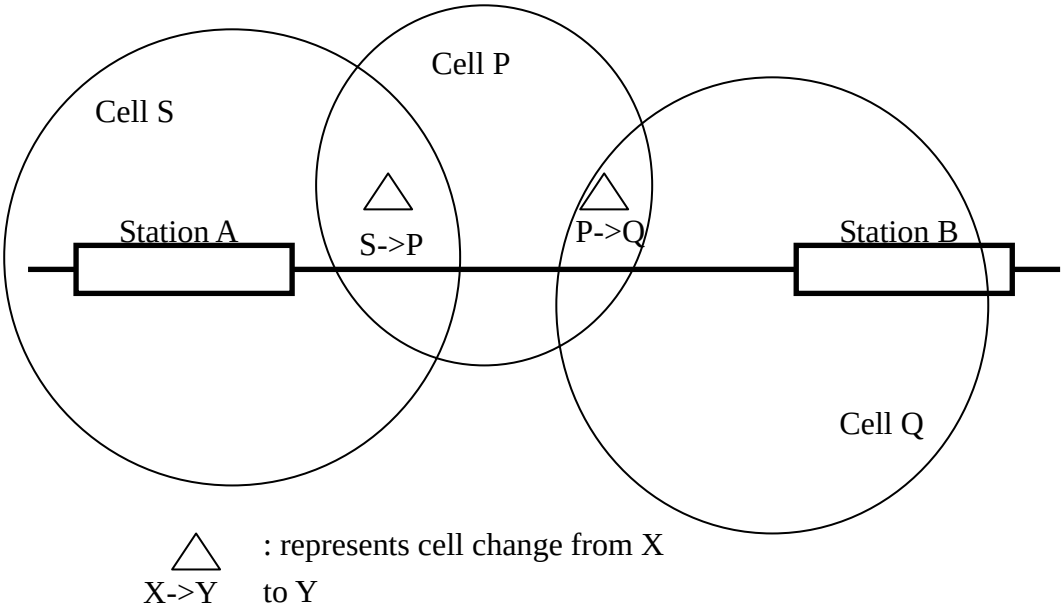


Figure 7.2: Principle of the experiment: records all the cell change event for the future use in developing our LBS system

7.3 Results and Statistics

This time, we also performed the experiment for both telcos, SmarTone and Peoples, on part of the MTR rail and KCR rail. The experimental data are tabulated as follows (left column represents the station transition, while the right represents the new cell entered during the journey) :

For Peoples:

Data for MTR:

Po Lam - Hang Hau	19061
Hang Hau - Tseung Kwan O	19051
Tseung Kwan O - Tiu Keng Leng	19041
Tiu Keng Leng - Yau Tong	19031
Yau Tong - Quarry Bay	9161

Quarry Bay - North Point	9231
Sheung Wan - Central	9031
Central - Admiralty	9011
Adimralty - Wan Chai	9371
Wan Chai - Causeway Bay	9021
Causeway Bay - Tin Hau	9321
Tin Hau - Fortress Hill	9081
Fortress Hill - North Point	9201
North Point - Quarry Bay	9211
Quarry Bay - Tai Koo	9231
Tai Koo - Sai Wan Ho	9291
Sai Wan Ho - Shau Kei Wan	9241
Shau Kei Wan - Heng Fa Chuen	9251
Heng Fa Chuen - Chai Wan	9131, {1381, 10713, 10712, 1381, 1401}
Yau Tong - Lam Tin	9161
Lam Tin - Kwun Tong	{842, 841, 1811
Kwun Tong - Ngau Tau Kok	1012, 931, 933, 3561
Ngau Tau Kok- Kowloon Bay	811, 3562, 11462, 11463, 2112
Kowloon Bay - Choi Hung	11451, 11452, 12131,} 9051
Choi Hung - Diamond Hill	9061
Diamond Hill - Wong Tai Sin	9381
Wong Tai Sin - Lok Fu	9171
Lok Fu - Kowloon Tong	9111
Kowloon Tong - Shek Kip Mei	9271
Shek Kip Mei - Prince Edward	9221
Prince Edward - Mong Kok	9192
Mong Kok - Yau Ma Tei	9391
Yau Mai Tei - Jordan	9101
Jordan - Tsim Sha Tsui	9341
Tsim Sha Tsui - Admiralty	9011

Note: CI in parenthesis represents cells in open area, other cells are in closed area. LAI in closed area is 120, while that in open area is 150

Table 7.1: Experimental results for MTR using Peoples

Data for KCR:

Kowloon Tong - Tai Wai	10203, 4293, 4091, {949, 9481, 661}
Tai Wai - Sha Tin	4511, 11571, 11572, 11573, 4101, 10202
Sha Tin - Fo Tan	4191, 11121, 11123
Fo Tan - University	2443, 10871, 3263, 4782, 10582, 10581, 5531, 4131, 4124

Note: CIs in parenthesis are in tunnel, others are in open environment. LAI in open area is 140, while it is 110 inside the tunnel.

Table 7.2: Experimental results for KCR using Peoples

For SmarTone:

Data for MTR:

Po Lam - Hang Hau	13051
Hang Hau - Tseung Kwan O	13041
Tseung Kwan O - Tiu Keng Leng	13031
Tiu Keng Leng - Yau Tong	13021
Yau Tong - Quarry Bay	13012
Quarry Bay - North Point	4921
Central - Admiralty	4942
Adimralty - Wan Chai	4941
Wan Chai - Causeway Bay	4934
Causeway Bay - Tin Hau	4935
Tin Hau - Fortress Hill	4936
Fortress Hill - North Point	4937
North Point - Quarry Bay	3438
Yau Tong - Lam Tin	42502
Lam Tin - Kwun Tong	12813, 42503, 12812, 42502
Kwun Tong - Ngau Tau Kok	4071, 43352, 40481, 20113, 43231, 43305, 12812
Ngau Tau Kok- Kowloon Bay	20382, 20001, 43352, 4071
Kowloon Bay - Choi Hung	41257, 43611, 9286, 43611, 41256, 4911
Choi Hung - Diamond Hill	4912

Diamond Hill - Wong Tai Sin	4913
Wong Tai Sin - Lok Fu	4914
Lok Fu - Kowloon Tong	81141
Kowloon Tong - Shek Kip Mei	81131
Prince Edward - Mong Kok	4961
Mong Kok - Yau Ma Tei	4962
Yau Mai Tei - Jordan	4963
Jordan - Tsim Sha Tsui	4964
Tsim Sha Tsui - Admiralty	4941
Admiralty - Central	4942

Table 7.3: Experimental results for MTR using SmarTone

For KCR:

Kowloon Tong - Tai Wai	61531, 682, 673, 5793, 30701, 31423, 4243, 4242, 8326
Tai Wai - Sha Tin	30766, 31422, 31983, 203, 30741, 33005, 31981, 8161
Sha Tin - Fo Tan	32611, 18431, 32611, 32353, 33232, 33231, 32042, 7482, 33135
Fo Tan - University	7353, 8341, 32041, 7481, 37513, 33553, 33731, 26391, 61511

Table 7.4: Experimental results for KCR using SmarTone

We discovered several observations associated with GSM cell change in MTR and KCR:

1. For underground stations, there is one particular GSM cell covering each station. Therefore, there is one and only one cell change detected in between any two nearby underground MTR stations. This facilitates the development of LBS since we can clearly identify the location (at which station) we are currently at.

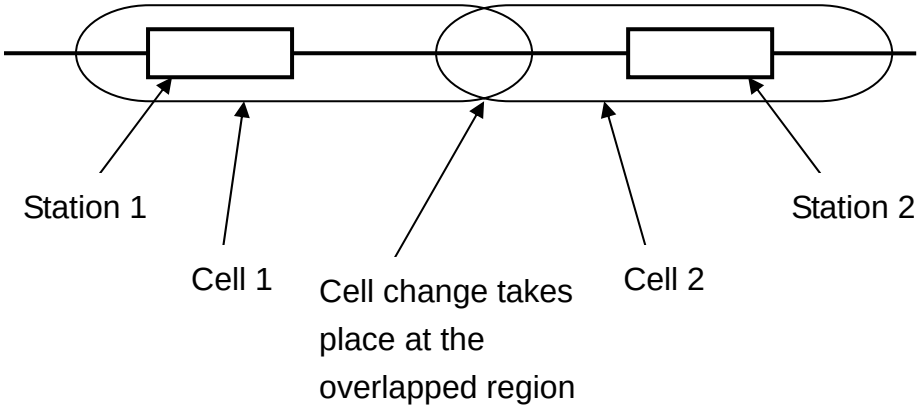


Figure 7.3: In tunnel or underground area, only one cell covers one station.

2. For station that is in open environment, for example Kwun Tong and Chai Wan, several cell changes are detected during a journey from one station to another nearby station. This is because in an open environment, GSM cells outside the MTR station are reachable from the train, so more that one cell can be reached by mobile phones when the train travels in open environment. For example, while traveling from Kwun Tong to Ngau Tau Kok, cell identifier changes from 1012 to 931, from 931 to 933 and finally reached 3561.

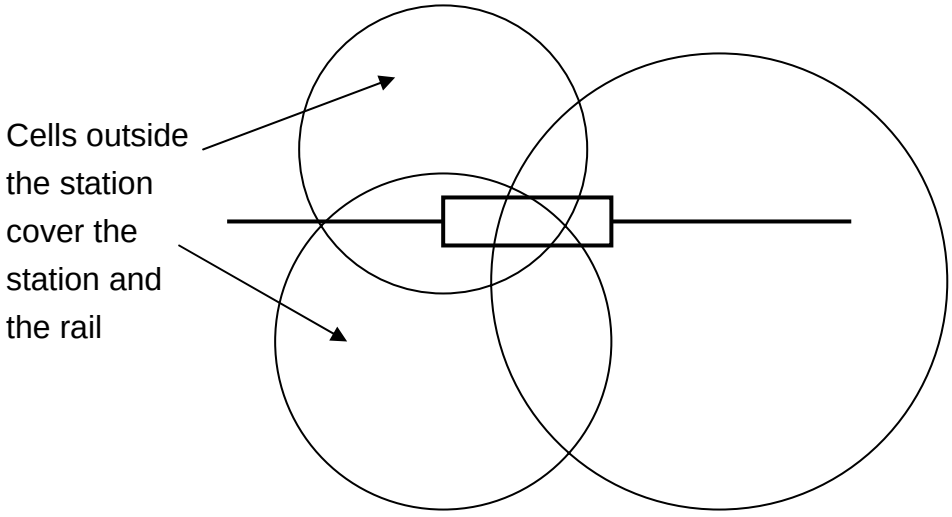


Figure 7.4: In open area, multiple cells may be detected.

3. When the train travels at constant speed, inside a GSM cell, received signal strength is about 100%. However, when the train starts to accelerate or decelerate, the received signal strength drops. It rises to 100% when the speed approaches to a constant. This could be resulted from Doppler effect. When the traveling speed changes, the apparent frequency emitted by GSM antenna varies, thus the mobile phone cannot receive the desired frequency well. Only when the traveling speed becomes steady will the apparent frequency become steady, and makes the signal strength return to 100%.

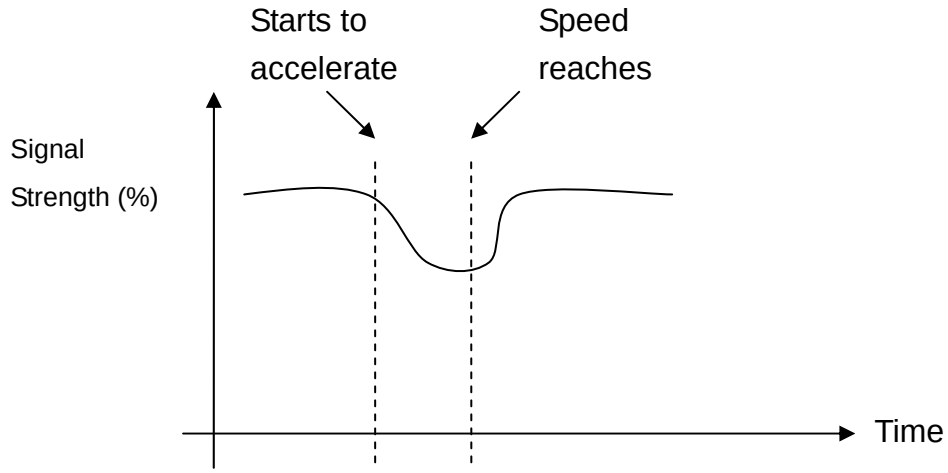


Figure 7.5: Signal strength drops when the train accelerates

4. Location area identity changes when entering or exiting a closed environment (tunnel or underground). This may indicate that GSM cells in closed environment are specially deployed, so as to let users within these places to access to the GSM network. For example, LAI change from 120 to 140 while traveling from Shau Kei Wan (underground) to Heng Fa Chuen (ground level).

7.4 Accuracy Measurement

After looking into the cell coverage in the 1-dimensional space (MTR and KCR in this project), we performed an experiment to investigate the accuracy of our LBS approach used in this space. In this experiment, we tried to measure the time needed for the train to get stopped at a particular station after our program detected a cell change event representing that we are entering that particular station.

Since the size of overlapping of cell is not negligible, a cell change event may happen at any point within the overlapping region. For example in figure 7.6, the route g to h passes through a cell overlap region. A cell change event may take place at any point inside the overlap region, for example point U or V. The place at which cell change take place is very dependent on the environmental condition, which affects the signal strength received by the mobile phone.

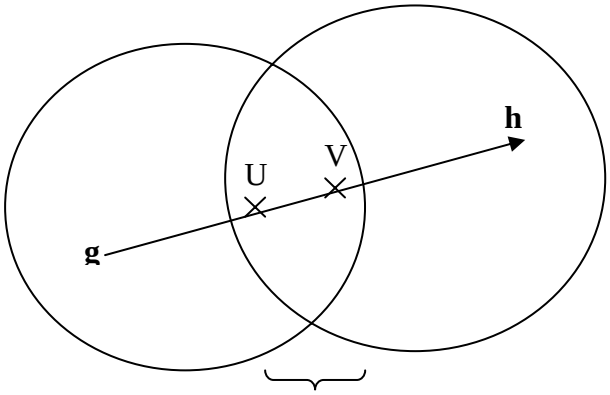


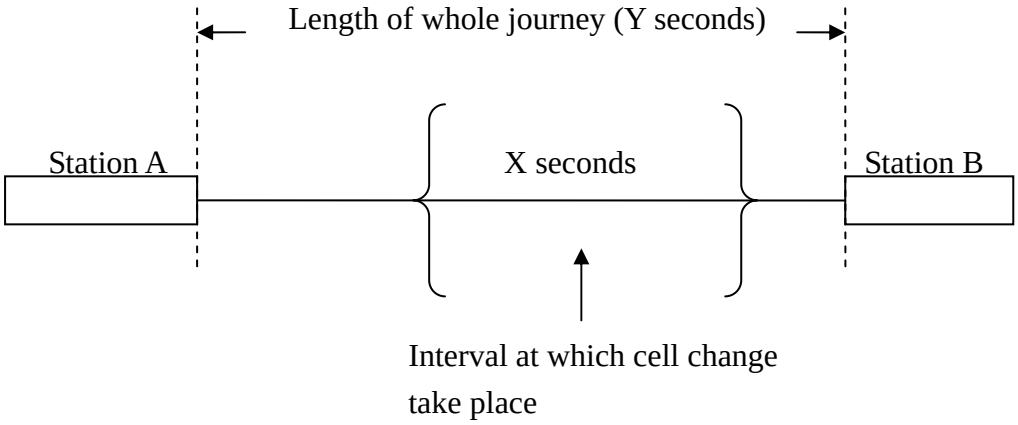
Figure 7.6: Cell change may take place at any point inside the overlap region

By finding out the portion of time in which cell change event may occur in the journey from one station to another nearby station, we can estimate the accuracy of using this LBS approach in the MTR.

We used time as our measurement metric since the speed of an MTR varies greatly in a journey, which make it hard to find out the exact length of the journey in term of distance.

We first estimate the average time needed for a train to go from one

station to another station, and then we find out the interval of time at which cell change events occur by repeating the experiment several times..



If the interval at which cell change may take place is small compared with the length of whole journey, then the result is said to be quite accurate since all the cell change events happen nearly at the same point in the journey.

Table 7.6 shows the result of the experiment we done at some MTR station. All the data are taken in the metric “second”. We performed the experiment at different dates so as to get a more general result, since the environmental condition varies from day to day.

The column “Variation of time” shows the accuracy we obtained. The term “Variation of time” here we mean the interval of time at which cell change occur within the whole journey. We find out that most of the results we got are quite accurate. They are mostly around 10-15% of the length in the whole journey.

From	To	Average Total Time of Journey	11/17/2003	11/18/2003	11/20/2003	11/21/2003	11/22/2003	11/24/2003	11/25/2003	Variation of Time
Po Lam	Hang Hau	83.4	32.0	34.5	35.3	39.1	38.1	36.8	36.4	0.085
Hang Hau	Tseung Kwan O	106.3	45.0	56.4	71.5*	49.0	61.0	49.6	57.3	0.112
Tseung Kwan O	Tiu Keng Leng	74.0	21.0	29.2	30.5	16.8	31.1	18.4	X	0.136
Tiu Keng Leng	Yau Tong	136.8	42.1	62.7*	43.0	46.1	47.9	41.5	39.0	0.086
Yau Tong	Lam Tin	103.4	1.0	X	4.5	X	X	1.6	1.8	0.034
Choi Hung	Diamond Hill	79.8	34.3	31.4	38.6	X	X	27.1	31.5	0.144
Diamond Hill	Wong Tai Sin	74.5	16.8	22.2	23.3	X	X	21.4	21.8	0.087
Lok Fu	Kowloon Tong	97.2	25.0	60.0*	X	X	X	16.6	47.4	0.317
Kowloon Tong	Lok Fu	74.5	18.5	13.2	14.9	7.1	X	18.9	14.7	0.158
Wong Tai Sin	Diamond Hill	60.0	28.7	28.9	30.0	32.1	36.2	15.6	X	0.343
Diamond Hill	Choi Hung	83.8	36.8	36.7	32.4	33.9	33.3	32.1	35.4	0.056
Lam Tin	Yau Tong	80.2	59.5	40.0	44.1	44.0	45.3	41.7	43.1	0.243
Yau Tong	Tiu Keng Leng	150.7	78.0	81.3	70.6	91.6	88.3	91.3	84.3	0.139
Tiu Keng Leng	Tseung Kwan O	77.4	27.0	23.2	28.3	29.9	28.3	29.9	31.1	0.102
Tseung Kwan O	Hang Hau	58.5	46.3	31.8	37.7	39.2	46.7	37.5	X	0.254
Hang Hau	Po Lam	87.8	X	22.7	X	38.1	35.5	30.9	25.4	0.175

Note: X: Not recorded or MTR Traveller cannot detect the current station

*: The train travelled at an abnormal speed (e.g. stopping the train on the way between two stations)

Average speed of Hong Kong MTR should be 33km/h according to the webmaster of Hong Kong Rail Engineering Centre

Table 7.5: Statistics on the time needed to reach a station after our program prompts

7.5 Conclusion to the Experiments

The experiment shows that LBS in 1-dimensional space are feasible. No matter in closed environment or in open environment, we can easily determine which region or station we are at in the whole 1D route. For in close area, only one cell covering a station gives us a clear-cut of the realm covered by a particular base station, thus we are sure to be in a specific region without uncertainty. For in open area, several cells may be accessible to the route during a journey. This gives us higher accuracy of determining the location we are going to. For example, there are five cells (A to E) in between the journey from g to h. If we experience cell change events A->B, B->C and C->D, then we are quite sure that we are going to h before we exactly reach h. This extra information helps us to predict our destination.

LBS in 1-dimensional space can be used not only in MTR, KCR and buses. Everything that has a path can utilize this service. For example we can implement a system for passenger to use, so that the system will prompt the passenger when he should get off the bus, or prompt him when it is time to change for other means of transport. Furthermore, we can implement a system that warns the driver when he is entering a region where there would be speed inspection so that he could reduce his speed in time. Lastly, we could also use it to calculate a more accurate location in 2D space by the “history” of his path taken.

7.6 MTR Traveller

With the 1-dimensional data in MTR and KCR, we implemented a program named MTR Traveller. As its name implies, it is an electronic travel agent used in MTR (and part of KCR). It is still a prototype of the whole system.

7.6.1 The Function

In MTR Traveller, we have implemented function that tells the user his current location in the journey. It also displays the corresponding map of the railway for the user to refer to.

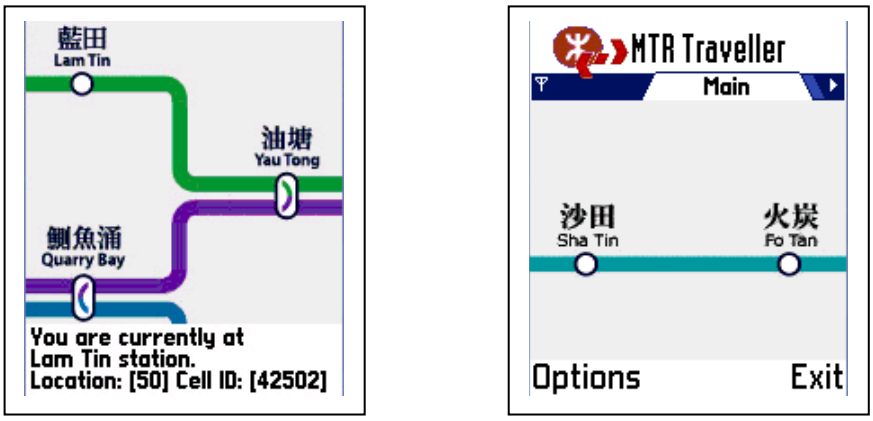


Figure 7.6: Screenshots for MTR Traveller, an electronic travel agent used in MTR and KCR

For example, when we are traveling from Causeway Bay to Wan Chai, the system will prompt the user that he is going from Causeway Bay to Wan Chai. It also displays the cellular information onto the screen. We could add more useful functions onto this program. For example, we could add a function that find the shortest path for going from one station to another station; we could make the program more informative by adding other public transport, like bus, that a user can change at that particular station; we could also add some information about the sightseeing sites nearby that station for tourist to go. All these can easily be implemented onto MTR Traveler. And we suppose we would do so in the next semester.

7.6.2 Program Architecture and Concept

This program is conceptually simple, though not easy to implement. In this program, there are three main components: the location sensor, the database controller and the graphical display unit.

The location sensor is in fact the extension of GSM Status. It detects the cell change event in every one second interval. If a cell change event is detected, it makes a query to the database controller for the current location using the current and the new LAI and CI pair. The database controller, upon receiving the query, returns with the names and GUI positions of the source station and destination station. These information are then displayed to the users.

We store all the information, including the LAI CI pairs, station name, and GUI positions into a database bundled with the Symbian OS. The DBMS inside supports simple data query only, no aggregate function is supported.

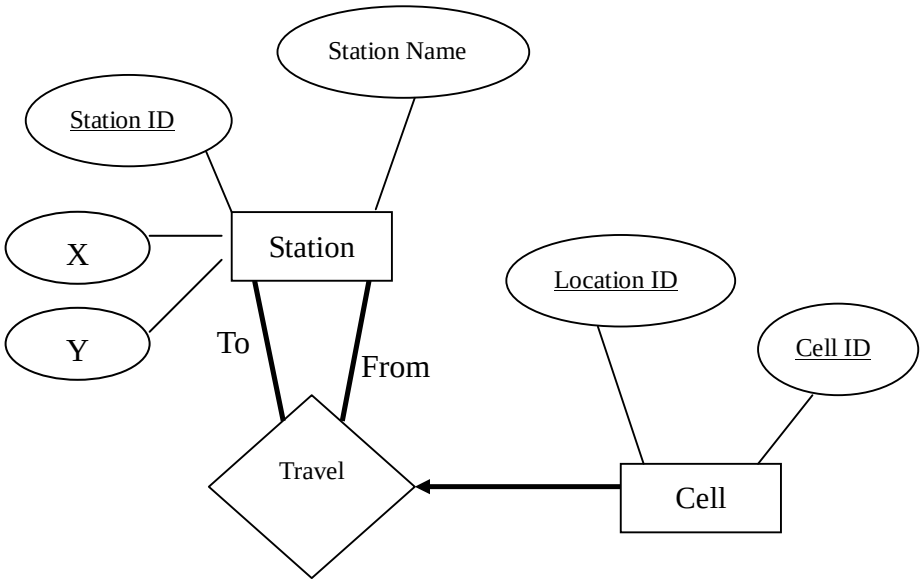


Figure 7.7: Entity-relationship diagram for the database we used

We used two tables to store the data for the transition from one cell to another cell during a journey from one station to another station. In the tables we record the data of stations, new cell entered and the transition information, which includes station IDs of the two stations, and the new cell discovered.

Schema of the tables we used:
Station (Station ID, Station Name)
Transition (Station ID 1, Station ID 2, Location ID, Cell ID, X, Y)

The GUI display in fact only displays one bitmap for the whole program. This bitmap includes all the necessary graphics for each station we did experiment. The GUI position required from the database indicates which portion of the bitmap is to be displayed. The GUI position is in fact an (X,Y) coordinate, which is used as the starting point (top-left corner) of the bitmap to be displayed.

8. The Middleware

8.1 Situation

From previous chapter we conclude that our approach towards building location-based application is feasible and of nice accuracy in the one dimensional space. As the world is migrating to the wireless era, the demand of mobile location-based services emerges. Although building location-based application using our approach is nice, the development of location-based services can be quite cumbersome since developers need to collect cell information, analyze and process collected information before building the real application. In many cases, building the application can also be quite difficult. It is because developers need to dig into the system API in order to retrieve location information. They also need to tailor-made each application so that they fit different purposes.

8.2 Motivation

As most company prefer to shorten system development time, we would like to provide developers developing location-based services using mobile phones with Symbian OS a middleware, which includes a set of application programming interface (API) together with several toolkits, so as to facilitate system development.

In our middleware, the API set should allow developers to perform most type of location-based functions. These functions may include retrieving current location information, keeping track of location change event, performing a particular action upon entering a set of specified GSM cell and also searching for a nearest point of interest by reading a location definition file.

8.3 Location-based Application Development Process

The following flow diagram shows the general flow in developing a location-based application:

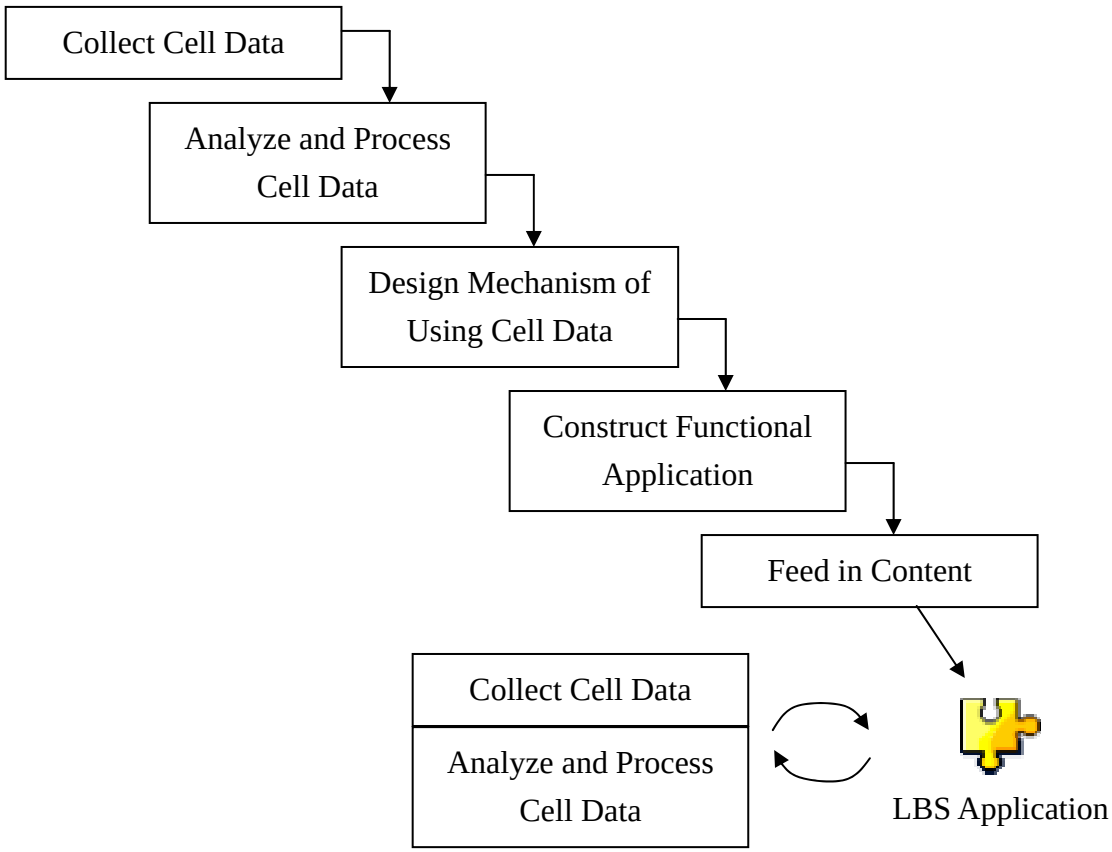


Figure 8.1: LBS Development Process

The development process is explained as follow:

Cell Data Collection:
Cell information is the basic element for the positioning method to work. Developers have to take cell data information by traveling the region of interest, such as MTR route for MTR Traveller, and gather the data.

Cell Data Analysis and Processing:
Raw cell data collected should be analyzed before importing into the application. For example, user may remove cell change duplicates and adjust the sequence of cell pair ordering. Moreover, usually data from different network operators are taken in different time. Developer is responsible to combine all these data together into single data file so that application itself does not need to identify which network operators the user rely on (independent of network operators subscribed).

Mechanism Design:
A mechanism of using cell data, such as cell change detection, has to be implemented programmatically. This is necessary prior to building an application.

Functional Application Construction and Content Import:
Developers can start designing their applications. The word “functional” means that the application can accept content from a dedicated source (e.g. local data file or network). This separates the tasks of software development and content/service offering.

It should be noticed that cell data collection, followed by analysis, should be done regularly as network operators may reconfigure their base station settings from time-to-time.

Therefore, besides giving developers a way to perform handset positioning, complication at each step should be reduced in order to chop down the development effort.

8.4 Middleware as a Solution

Nowadays, software companies usually ships their products, such as library, with a set of toolkits, forming an easy-to-use development environment. Typical examples are Microsoft Visual Studio, NetBeans, Symbian SDK and Emulator. We found that similar approach can be taken in our case – a middleware, consisting of a well-defined APIs and a set of tools for developers to build, and also maintain, a LBS application in a time-saving manner. This can be achieved by:

Encapsulation:
Low level function calls can be invisible from developers by wrapping up them into library. A well-defined application programming interface, API, would be provided to developers to work with. Besides, the whole GSM cell ID handling can also be hidden through a set of toolkits.

Automation:
Considering the whole LBS development process using cell ID,

developers have to collect cell data, process them, build a LBS application for a particular purpose and all of them should be done manually. For example, in tradition, cell data are collected by reading data from phones or linking with extra devices, like PDAs, as mentioned in the section of cell data collection (Chapter 5). There should be some toolkits to automate some of the work, or at least reduce manual processing.

Layering:
Interfaces and tools provided should be arranged in layer approach such that developers can view the underlying implementation as a black box when interacting with a particular interface or tool. Developers of different concerns can start their work at corresponding ‘layer’.

8.5 The Architecture

In order to perform the task of building an location-based service system in a simpler and faster manner, our system must include several components: an API set, tools that help to collect, process and analyze cell information. We also need an application that help the developer in setting the relative distance among those point of interest, so that there is a distance definition file to be referred when searching for the nearest point of interest. Last but not least, we need an application generator which generates location-based service application automatically with parameters set by developers.

The following figure shows the architecture of our middleware.

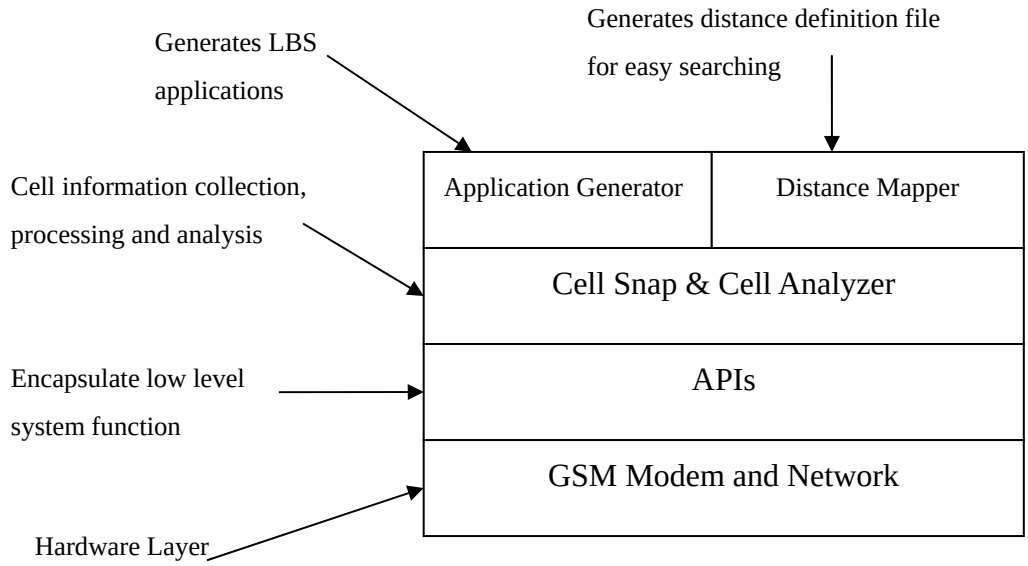


Figure 8.2: Overview of the development kit architecture

As seen from the above figure, the development kit contains three different layers built on top of the hardware layer.

Just above the hardware layer comes the API layer. This layer APIs which encapsulate low level system call, so that it allows developers to retrieve and manipulate location information without going through complex steps in invoking those system calls.

Cell Snap and Cell Analyzer are used to collect and process cell information. Traditionally, there is no application written for developers to collect GSM cell information. Developing an application for merely collecting cell information can be time-consuming. Users usually have to remember all location and cell ID pairs together with their relative physical location in order to associate those ID pairs with their physical location. With Cell Snap and Cell Analyzer, developers are now able to collect location information in a handy way. Cell Analyzer helps to group and manipulate location information collected from different telecommunication companies.

At the top level there are AppGen and Distance Mapper. AppGen is a power application that allows automatic generation of location-based

mobile application. Developers can set several parameters such as what type of location-based service is going to be included, what action to be taken upon reaching some point of interest, which location change should be observed etc. This application allows easy generation of location-based application without having to write much code.

Distance Mapper helps to map collected location information to their physical location, and it generates a location definition file that will be read by the application generated by AppGen for providing the service of searching for nearest point of interest.

8.6 Some Definitions

Before investigating how the middleware provides necessary functions for developers, some definitions are introduced before hand.

8.6.1 Reference Point and Point of Interest

Location-based service usually focuses on a set of regions or points, such as building, shopping mall and tourist spot. A **point of interest**, or POI (PsOI for plural form) in short, represents a particular point on the map of which the LBS application would be aware. For instance, an interactive campus visitor application should have all buildings and canteens as points of interest.

However, as experimented before, GSM cell ID positioning method performs best on 1D path. This implies that the application should consider a path rather than the whole 2D map. Therefore, what the developers concern actually is the **reference point** on a path. Reference points are those points taken in cell data collection process on a predefined path.

To illustrate clearly the idea, an interactive tourist guide along a bus route is taken as an example. The application regards tourist spots interested as PsOI. However, such application may be designed for a particular bus route (or multiple bus routes), so bus stops involved can be considered as reference points as shown below:

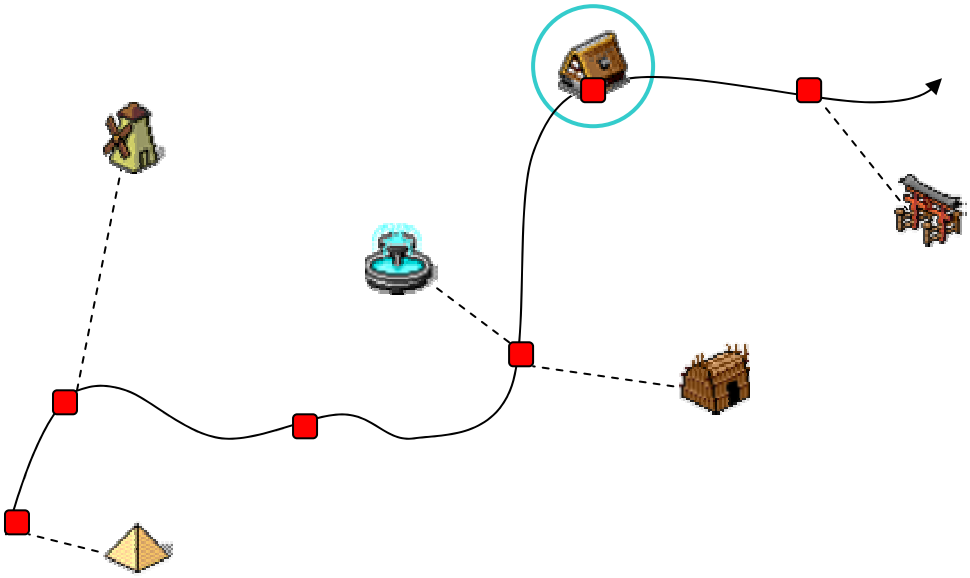


Figure 8.3: Reference Points and POIs of an Interactive Tourist Guide

Each POI (i.e. tourist spot) has a corresponding reference point (i.e. bus stop) associated along the bus route. Also, POIs may also lie on the route so it is also a reference point (indicated by the circle).

8.6.2 Location Definition and Distance Mapping

As mentioned in the previous section, user may need to know the nearest target region, such as fast-food restaurant nearby. However, purely with cell data, the application is not able to identify which cell is near or far away from current cell without giving also distance or geometric information. Therefore, it may require an extra step, called location definition, to associate the cell into to physical location, like this:

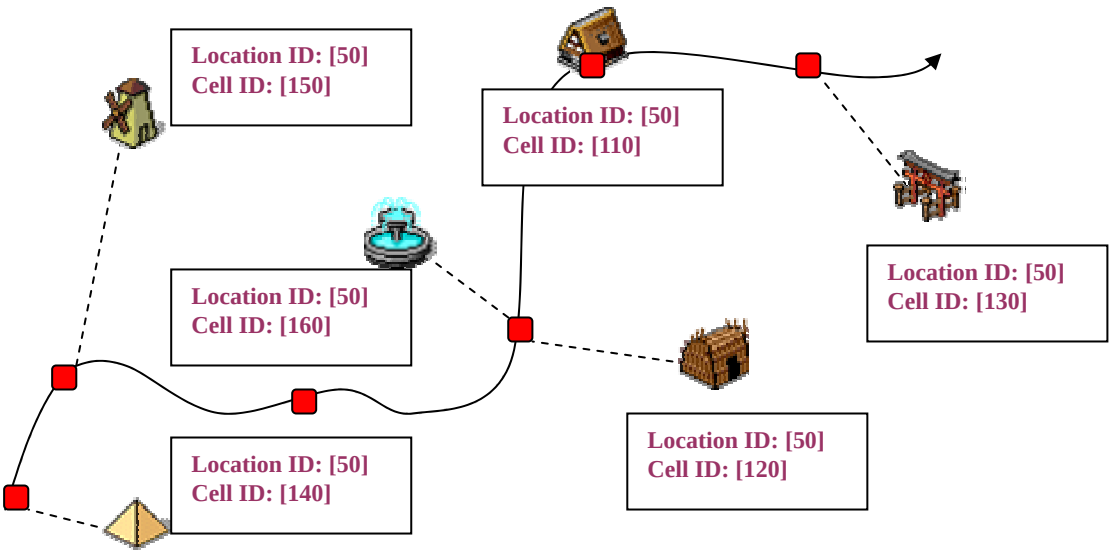


Figure 8.4: Mapping Cell with Geometrical Points on a Reference Map

As a result, the distance between cells can be calculated – distance mapping. It should be noted that what developers do is to approximate the cell because the actual location of a cell centre is unknown. However, this does not contribute a great problem as distances are used to find nearest point of interest but not precisely and accurately identify a location.

9. Application Programming Interface

There are existing system calls that allows developers to retrieve current location information via GSM Modem. However, calling these system calls can be quite troublesome. We need to connect to the GSM modem, creating an active object for requesting those information and many more. However all these operation seems to be a must-do when implementing location-based applications. In order to simplify the operation of generating system calls in retrieving and manipulating location information, we designed a set of APIs which facilitates the development of location-based application.

9.1 API Design Principle

In order to design good APIs that are complete and easy to use, we adhered to several design principle in designing our APIs.

9.1.1 Packaging

In a complete set of API there are usually a large number of functions. These functions may serve for different goals. These API functions can thus be grouped into different sub groups. Within each sub group, all functions serve for a similar goal. For example, a function that connect to the GSM modem can be grouped with functions that retrieve Cell ID and Location ID since they both serve for getting location information. By grouping functions into groups with the same goal can make the API more clear and easy to use. Users only need to know what goal they want to achieve and then use that particular package of API functions.

9.1.2 Naming Convention

Different programming languages have their own naming convention. For example Visual C++ uses the Hungarian notion; Java uses capital letter for the first character in their class name while methods must start with a lower case. These naming conventions, although seems simple and unnecessary, play an important role in giving information to end users. Therefore we followed the naming convention in designing the

API set. We followed the naming convention of Symbian. Some examples of naming convention in Symbian are: Classes that create objects in heap starts have a name starting with a character ‘C’; functions that may leave should end with a capital letter ‘L’ etc. For example, the following line:

CDesCArray* InsertWordL(TDesC& aWord)

represents a function that insert a word into a descriptor array and it may leave. aWord is the input parameter while this function output a pointer to CDesCArry which creates object in heap and therefore need cleanup after use.

9.1.3 Interfacing

As the API set becomes complex, sometimes we need to have a series of API calls so as to perform a particular task. In this case we may want to wrap series of API calls to a single API functions, providing users with the interface to use, while hiding internal operations from them. This can simplify the usage while making the API more convenient to use. This also make the API more robust since end users need to call less API functions to accomplish a task and minimized the chance of getting error.

9.1.4 Input Parameters and Return Values

The last consideration is the input parameters and return values. To make our API more general, we should use interface class as our input parameter of our function. Using an interface class instead of an implementation class make an API more general. For example, TPtr and TBuf are both derived class of TDesC. We would use TDesC as our function parameter instead of the former two because a function accepting TDesC means it can accept both TBuf andTDesC. Thus we increased the freedom to users to choose the input parameter type.

Contrast to input parameter, return value should be as specific as possible. This is because an implementation class usually are more

powerful than interface class since implementation class should have more helper functions which help to manipulate data in that object. Besides, users do not need to waste time performing type casting.

9.2 The API Structures

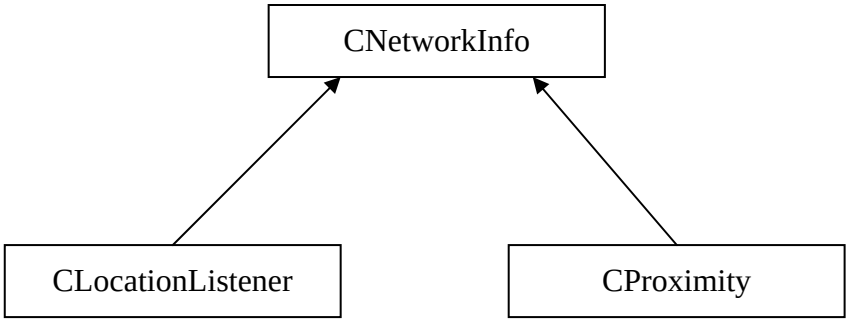


Figure 9.1: Class view of the API structure

As shown in figure 9.1, our API mainly consist of three main parts. One is CNetworkInfo, which helps to connect to the GSM modem, send information retrieval request and retrieve location information. CLocationListener keeps track of location change events and perform a specific action when specific cell change event take place. The last component, CProximity, searches for the nearest point of interest relative to the current location. These components will be explained in detail in the next sub chapter.

These three components should form a complete set for location-based service. In fact, after an in-depth investigation, we find that most location-based service can be divided into two types: one is searching for nearest point of interest, the other is keeping track of location change, then perform an action upon that cell change event occur.

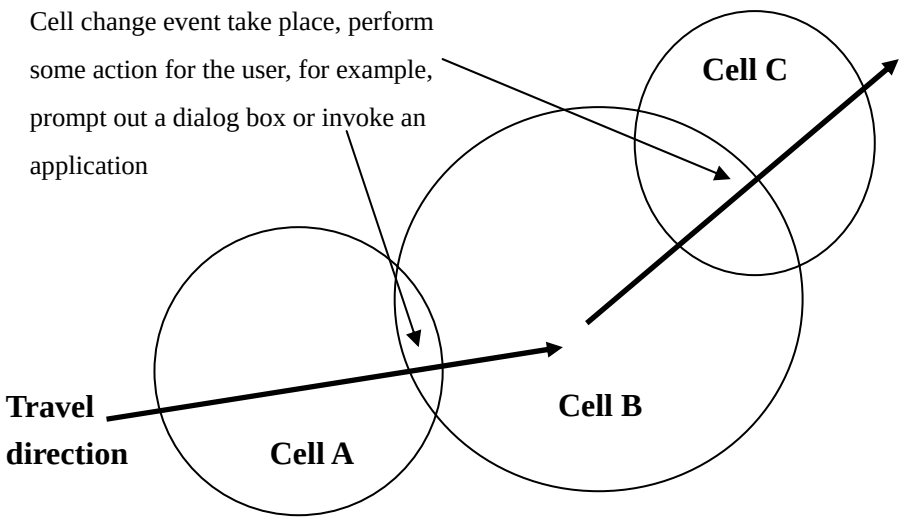


Figure 9.2 Perform an action upon specific cell change event occurs

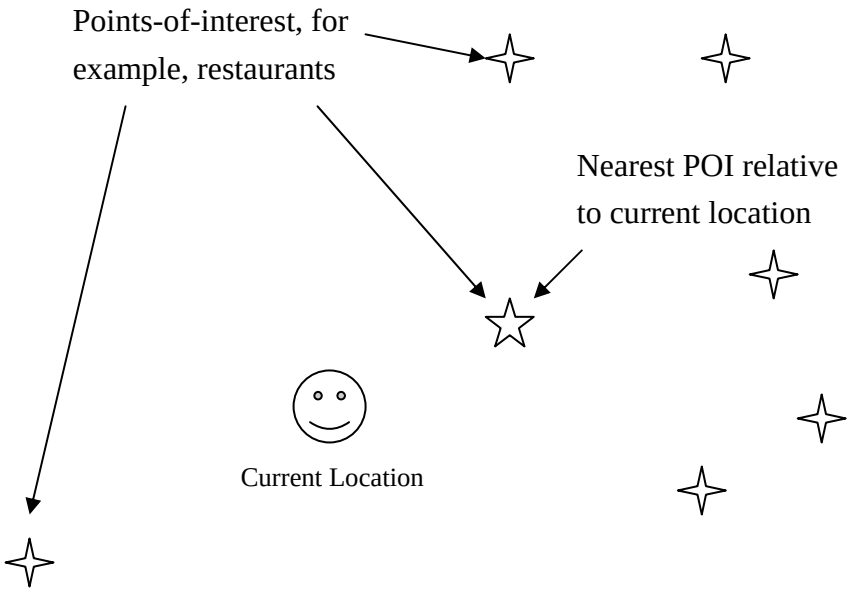


Figure 9.3: Searching for nearest POI relative to current location

9.2.1 CNetworkInfo

As mentioned before, CNetworkInfo is used to retrieve location information without the need for developer to perform low level system call. There are a number of functions in this class which can be used by developer while they are developing location-based application. Detailed specification for this API class can be found in the appendix section

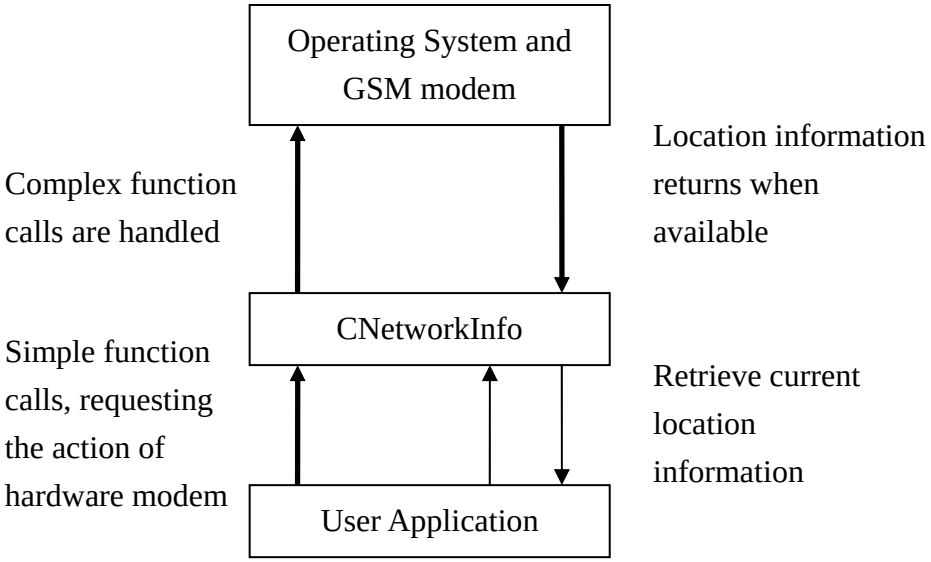


Figure 9.4: The role of CNetworkInfo

From the figure above, it clearly shows the role of CNetworkInfo. CNetworkInfo, once received user application request, sends out a request to the operating system for location information. The retrieval of location information is done in an asynchronous manner. Once a new set of information is available, it will be returned to CNetworkInfo. CNetworkInfo will store the most update location information and allow user application to retrieve by using simple function calls.

9.2.2 CLocationListener

One of the two most popular location-based services is to keep track of cell change events. CLoctionListener performs this task for developers.

CLocationListener makes use of CNetworkInfo in retrieving location

information as it needs to keep track of location change event. Since CLocationListener keeps track of location change event, so it knows whenever cell change occurs. In order to increase the flexibility for developers, a list of cell ID which the program is interested in should be supplied to it. By referring to the list, CLocationListener will be able to perform specific tasks when entering cells that are specified in the list.

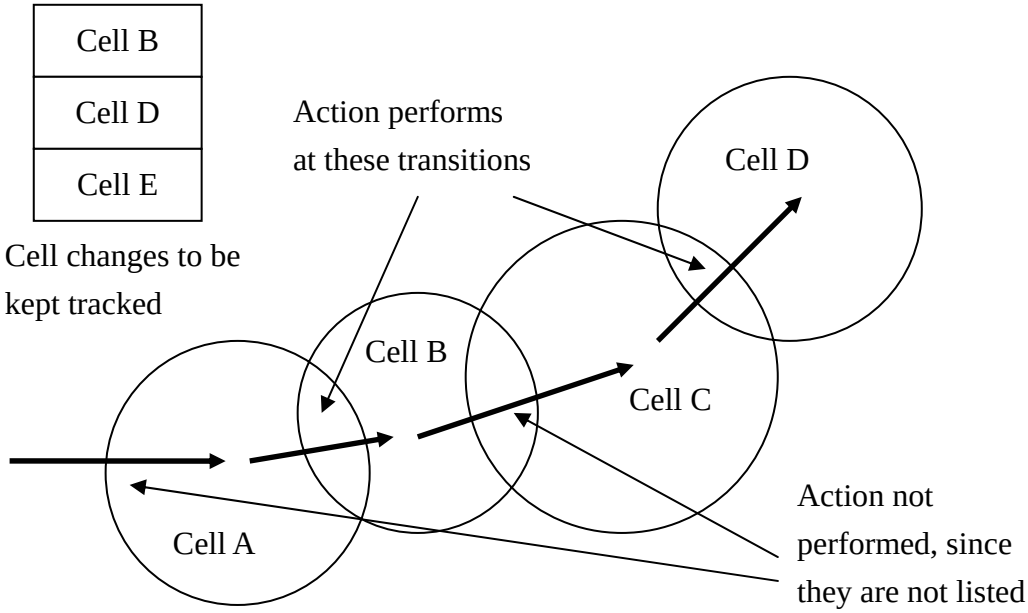
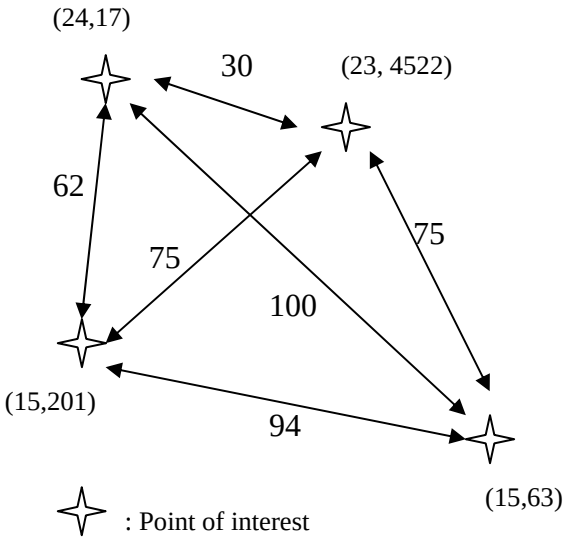


Figure 9.5: Principle of how CLocationListener works.

9.2.3 CProximity

This API class performs the task of searching for the nearest point of interest relative to the current location. Users have to supply a location definition file which defines the geographical proximity of all point of interest.




```
; Sample location definition file
4
24#17
23#4522
15#201
15#63
; start of distance definition
99999#30#62#100
30#99999#75#75
62#75#99999#94
100#75#94#99999
; end of distance definition
```

Figure 9.6: Physical distribution of four point of interest and its location definition file

The above figure shows the physical distribution of four point of interest and its location definition file. In the figure, stars represent point of interest, for example restaurant, bus stop, toilet etc. Numbers in bracket represent the location ID and cell ID pair the POI is located in; numbers on arrows represent physical distance between two POIs.

The generation of location definition file can be done manually or with the help of Distance Mapper, which is a part of our development tool kit. We will explain the use of tool kits in next chapter.

To have a brief conclusion on the API set, it simplifies the steps needed in developing location-based application while providing most common location-based functions.

10. Development Tools

The API set is ready to be used in creating location-based application. However we still need to collect and process location information. Furthermore, developers still have to write their pieces of code.

A complete middleware should help developers in developing their application in all aspect. Therefore, we created several development tools which take care of all steps in the creation process. These tools help developers in collecting and processing location information. There are also tools to map point of interest and generate source codes from location-based application. These tools fall into two main streams in the development process, 1) Automatic Cell Collection and Manipulation, 2) Automatic Program Generation.

10.1 Automatic Cell Collection and Manipulation

As introduced before, cell data collection and processing are the necessary steps in LBS development process. However, these steps often require manpower and, hence, become time-consuming, especially when they must be done regularly to suit the change of cell configuration made by network operators.

Also, developers have to deal with cell data from multiple network operators rather than a single one because the LBS application should support all operators with a single data file. At the same time, they have to do some analysis and filtering of collected raw data before use. As a result, the steps of combining cell data from different operators and processing all cell data at once are bothersome, especially when data size is huge.

For example, in Hong Kong, there are 6 operators, namely Orange, CSL, SmarTone, New World Mobility, Peoples and Sunday. Suppose that the LBS application is designed for a bus route with 15 bus stops and there are 8 cell changes in between each bus stop on average. Thus,

The number of cell ID involved = 6 x (15 - 1) x 8 = 672

Notice that this number is for single bus route. If the application concerns also multiple bus routes, or, for instance, links to other transport routes, the data size become large and it is impossible to process them by human in a short period. Therefore, tools for automation are of uttermost importance.

In the following sub section, the operations of Cell Snap and Cell Analyzer, part of the software development kit, are explained in details.

10.2 Cell Snap

In the past, cell data are collected manually or with devices like PDAs Cell Snap is designed for collecting cell data automatically by using the Symbian phone alone. This can be achieved now because Symbian OS offers developers with programming capability in accessing mobile phone hardware, including GSM modem, Bluetooth module and camera.

10.2.1 Operation of Cell Snap

Cell Snap is a Symbian program that allows:

1. storing cell change event automatically in a list of triples:
(Location ID, Cell ID, Time elapsed since program start)

2. capturing photo at reference points so that developers can identify actual locations in between cell data sequence.

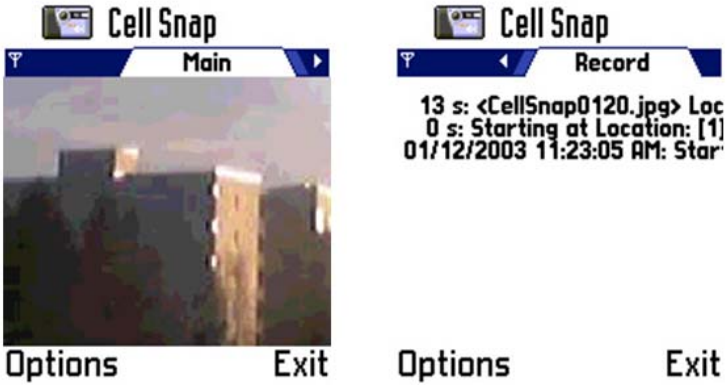


Figure 10.1: Cell Snap Screenshot – Photo Capture (Left) and Cell Data Record (Right)

The following diagram shows Cell Snap as a building block in the middleware. It captures cell data within the journey into a list of cell data and photos.

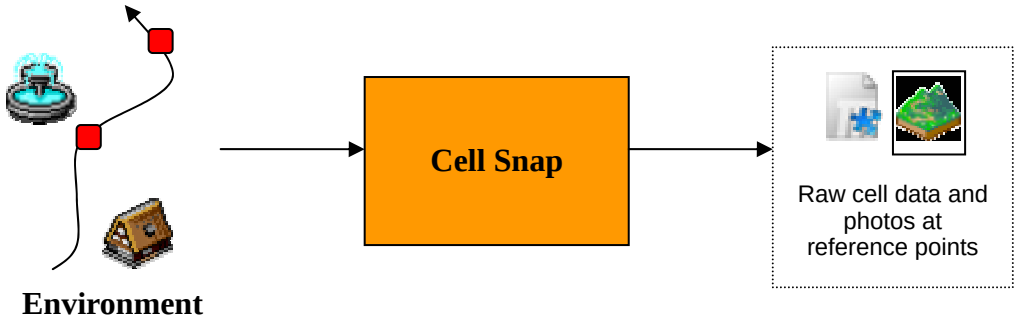


Figure 10.2: Cell Snap as a Building Block

10.2.2 Deficiencies of Cell Snap

Cell Snap is initially designed for small-scale cell data collection. This introduces certain problems:

1. Cell Snap is designed to be read by human, so the Cell Snap output data shows a list of raw cell change event log for human to read, like:

```
129 s: < CellSnap0002.jpg > Location: [ 130 ], Cell ID: [ 33731 ]
321 s: Location: [ 130 -> 130 ], Cell ID: [ 33731 -> 5812 ]
```

However, when there are more data, it is time-consuming to read this by human. Instead, it should be passed to machine to interpret. Besides, the API would require a list of cell data in specified format, so a conversion from this "human" text to concise text is required.

2. Photos and cell data files are stored separately. Developers should look at the content of data file and find the corresponding photo to map on their own. Therefore, it introduces complication to developers.

Also, Cell Snap only captures raw data. It is the responsibility of developers to edit the data by themselves. Therefore, this comes to the birth of Cell Analyzer.

10.3 Cell Analyzer

Cell Analyzer is a utility to edit cell data, combine data from all network operators and optimize the classification between reference points.

The following diagram shows the operation of Cell Analyzer:

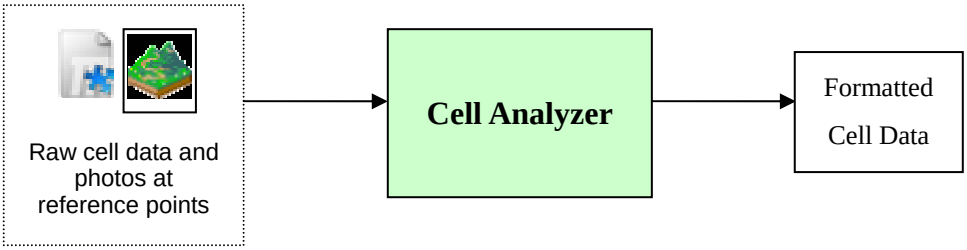


Figure 10.3: Cell Analyzer as a Building Block

10.3.1 Operations of Cell Analyzer

Cell Analyzer is written in Java and acts as a user-friendly tool for developers to process cell data. It provides following functions:

1. Data Format Transform:

Without any modification of data, it simply transforms raw cell data from Cell Snap into formatted data for further use (e.g. as an import file to construct location definition file through Distance Mapper).

2. Data Presentation:

Cell Snap data is not simply a list of cell data, but it can also be regarded as a tree, with reference points as parent nodes and cell data in between two reference points as child nodes. Besides, in the situation that multiple network operators' data involved in the same route, a 4-level tree structure can be obtained. Developer may find easier to understand the data through the user interface.

The following figure shows an example of a bus route project. It should be noticed that the leaf nodes are the cell data involved between reference points i and i+1. For example, the cells "Location ID [50] and Cell ID [160]" and "Location ID [50] and Cell ID [170]" are

involved in cell changes in between reference point 1 and reference point 2.

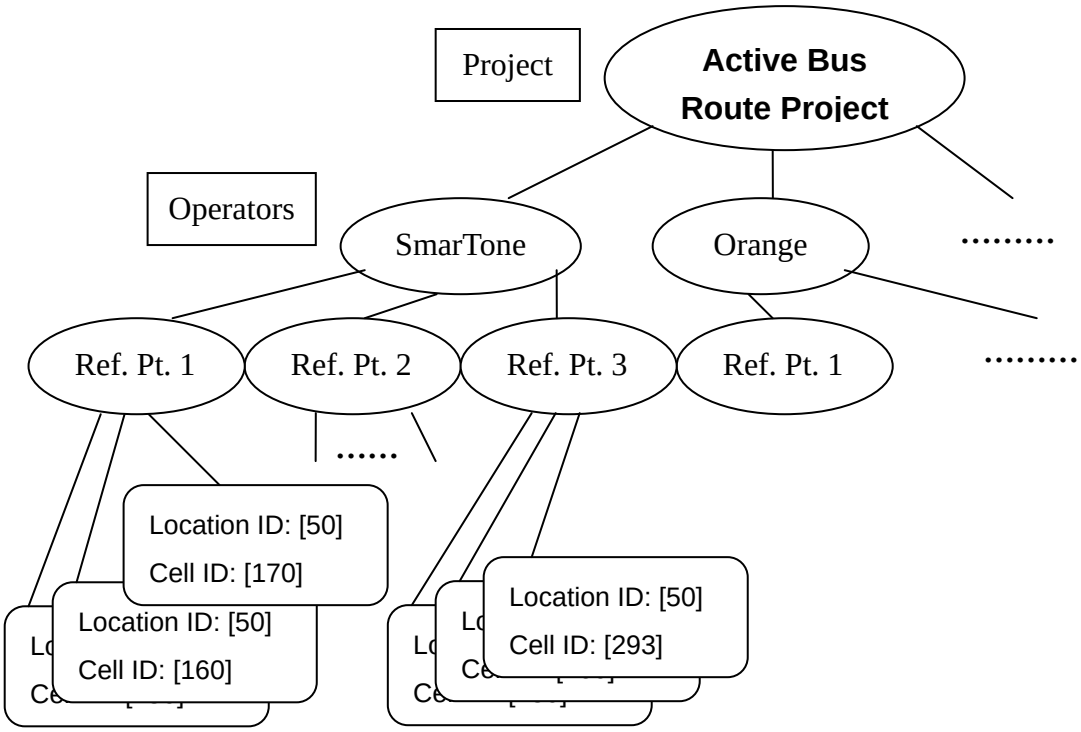


Figure 10.4: Cell Data in Tree Representation

3. Cell Duplicates Removal:

Consider the following diagram:

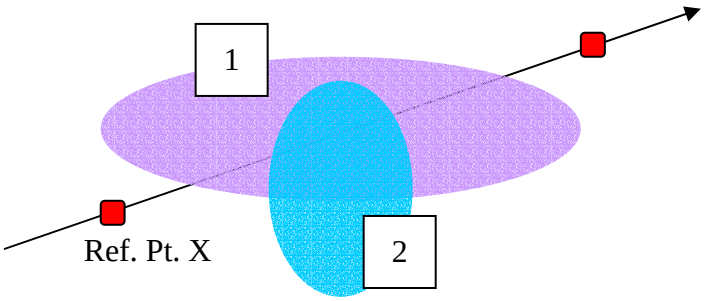


Figure 10.5: Cell Duplicates

There are two cells, cell 1 and 2, in between two reference points (indicated by the dots). Through the given path, the cell changes would be [Cell 1→ Cell 2→Cell 1] and result in Cell Snap data [X:Cell 1, Cell 2, Cell 1] (i.e. Reference point X contains Cell 1, Cell 2 and then Cell 1). However, the proposed location estimation method

only requires the existence of Cell 1 and Cell 2 in between transition these two reference points. Therefore, the second “Cell 1” can be removed. This can be done automatically by Cell Analyzer.

4. Intelligent Reference Point Classification:

The origin of this problem can be observed from building MTR cell data in open area. As mentioned before, in MTR Traveller implementation, cell data are classified as 1) station cell set and 2) transition cell set in order to identify the event of “You are in the station.” (station cells) and “You are in the way of station X to station Y” (transition cells). Similar optimization is taken in this middleware.

Consider the following situation in an open area.

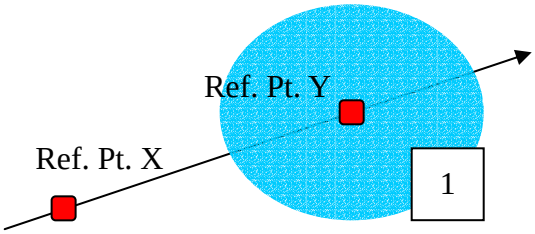


Figure 10.6: Situation in which Reference Point Classification is Needed

Both reference points X and Y contain a node of Cell 1. However, the expectation is, when mobile phone detects cell 1, it should represent the fact of “You are at reference point Y.”. Therefore, Cell 1 should be detached from reference point X in order to reflect this condition. Cell Analyzer would be able to classify which reference point a cell should attach to for open area.

5. Manual Editing:

Developers may manually adjust cell data, such as data removal, if necessary, rather than editing the import file from text editor.

6. Reference Point Pair-up for Multiple Operators:

Developers can define data from one of the operators as master data. As a result, all reference points from master data are considered as the main reference point set for the entire project. For reference points from other operators, they should be mapped to

corresponding reference points in master. Developers may choose data from operator with the smallest number of reference points as master data such that multiple reference points can be mapped to one master reference point. If developers only take some fixed points (like bus stops) as reference points, they may also perform “Auto Pair-up” to quickly map reference points among all network operators’ cell data. The cell pair-up interface is easy to use:

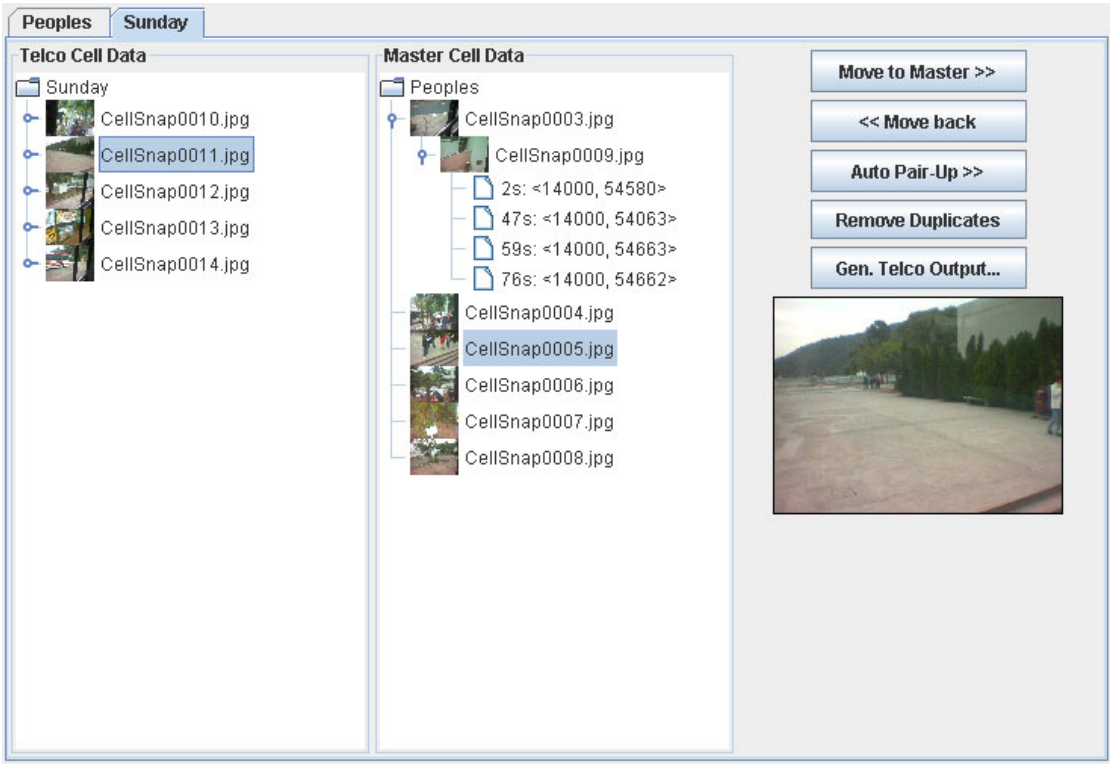


Figure 10.7: Cell Pair-up from Multiple Network Operators’ Data

10.3.2 Cell Analyzer Output

As mentioned before, Cell Analyzer would produce a formatted output for Distance Mapper, for assigning approximate locations for each cell, or AppGen, for LBS application generation.

Developer may choose to output a single file for individual operator or combined data for all operators’ data loaded.

10.4 Distance Mapper

Distance Mapper is one component of the development tool kits and is written in Visual C++. It is responsible for generating location definition file. A location definition file stores the approximate location of reference points. This file will be used by location-based application which provides the function of searching for the nearest point of interest relative to currently location.

10.4.1 Operation of Distance Mapper

The input file for Distance Mapper is the output of Cell Analyzer. Cell Analyzer format cell data collected by Cell Snap and generate a Formatted Cell Data File. Distance Mapper reads in the Formatted Cell Data File and let users to map those location information onto a physical map.

Figure 10.8: The Interface of Distance Mapper

The above figure shows the layout of Distance Mapper. On the left is the working space, a physical map can be loaded into the working space so as to map location information onto their physical location. There is a tool bar on the right. Users can load location information and physical map there. The top part of the tool bar is a list showing the location information retrieved from the formatted cell data file

outputted by Cell Analyzer. Users can map those data onto the map on the left-hand side.

10.4.2 Output of Distance Mapper

After mapping those location information onto the physical map, the program generates the location definition file which records the logical distance data among all point of interest.

The operation of generating the location definition file is as follow. After mapping all location information onto the physical map, the program calculates the relative logical distance in the pixel space. We simply use the distance equation in calculating the relative logical distance.

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

where x and y are the x- and y-coordinate of a location information in the pixel space. Since we only concern about which POI is closest to the current location, there is no point in calculating the exact physical distance. Relative logical distance can fulfill our need.

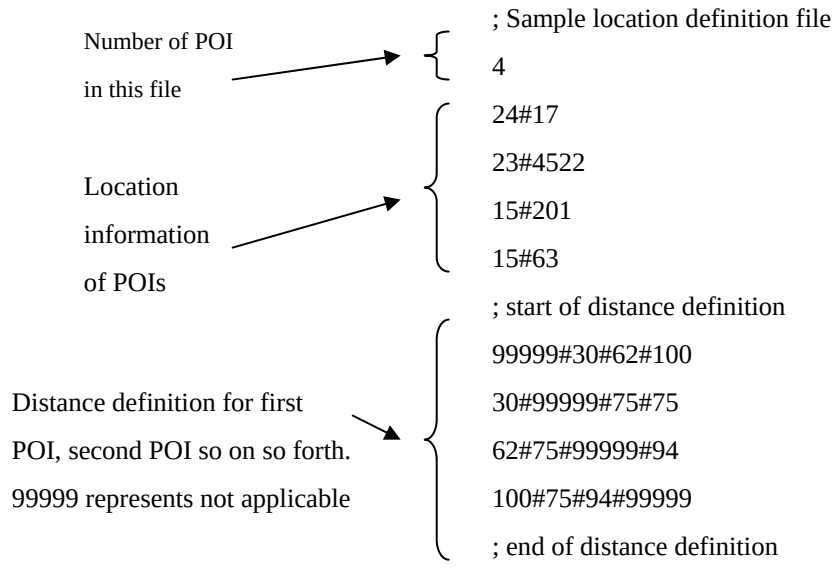


Figure 10.9: A sample location definition file generated by Distance Mapper and its explanation

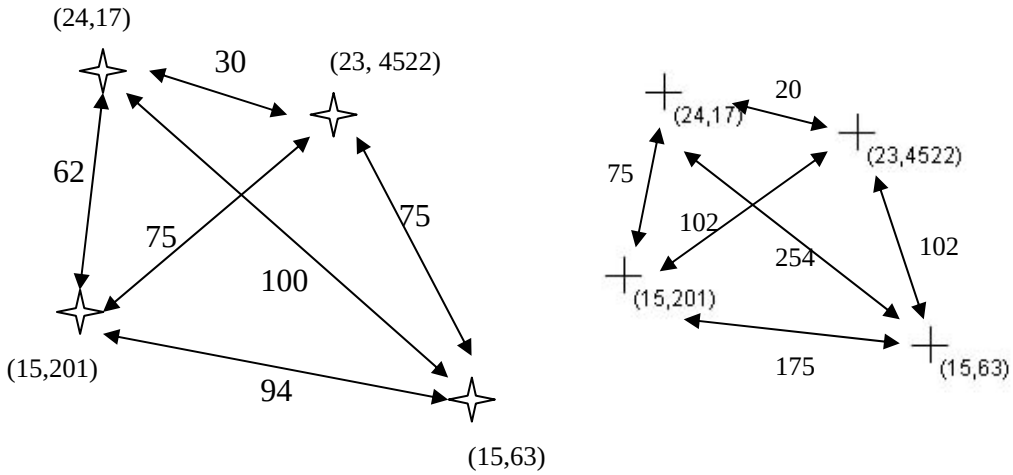


Figure 10.10: POIs distribution in (left) physical space and (right) pixel space.

The above figure illustrates the difference between exact physical distribution and logical distribution. The distributions do not need to be exactly the same. Both geometry and distance can vary, provided that the degree of separations among all POIs is preserved.

The output file will be used by location-based application for providing the search function

10.5 Automatic Application Generation

Besides manipulating location information, we also need an application generator that automatically generate a location-based application. AppGen is the program which handles this task.

AppGen is specifically designed for content builders who concentrate on content provided and actions performed when application users enters or leaves a region. As the name of AppGen implies, it generates source code of a multi-functional LBS application using the given GSM Cell ID positioning method, while AppGen users can edit the text, images, etc, prior to source code creation.

10.5.1 Operation of AppGen

AppGen refers to data from Cell Analyzer to define the reference points as follows:

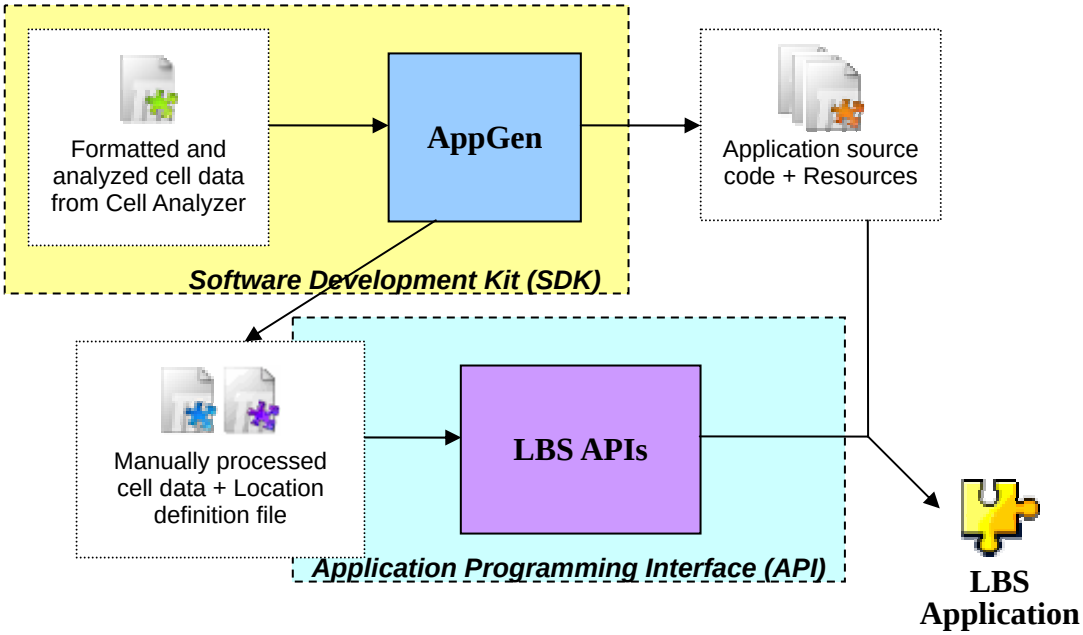


Figure 10.11: Role of AppGen in the LBS Application Development Flow

The role of AppGen is to generate source code for the application on the top of LBS APIs, so it requires cell data and location definition file for the code to work. At the same time, it also manages all resources involved in the application, including map, photos and icons.

As a result, content builders can modify the content while developers may consider the generated source code as the initial material to start build their own application.

10.5.2 Some Features in AppGen

Based on the assumption that content builders may not have deep knowledge in Symbian programming, AppGen aims to offer as much flexibility as possible to content builders (i.e. they can modify parameters and add new content in various formats) and ensure that changes provided should be closely relevant to content editing. Thus, AppGen provides three types of options for content builders, namely general options, reference point settings and point of interest (POI) settings.

10.5.2.1 Primary Features of the Application Generated

One should understand the functions that the application provided in order to decide the entire content needed. The application generated would cover as much as possible that a general LBS application can do, including:

1. To keep track of interested cell changes and report to user (e.g. entering a new station for MTR Traveller) and;

2. To let user to specify a destination and acknowledge the user when he/she reaches a reference point that associates with that destination (Those destinations are usually defined as point of interest (POI) here).

10.5.2.2 General Option in AppGen

The following screen shot shows part of the general options provided.

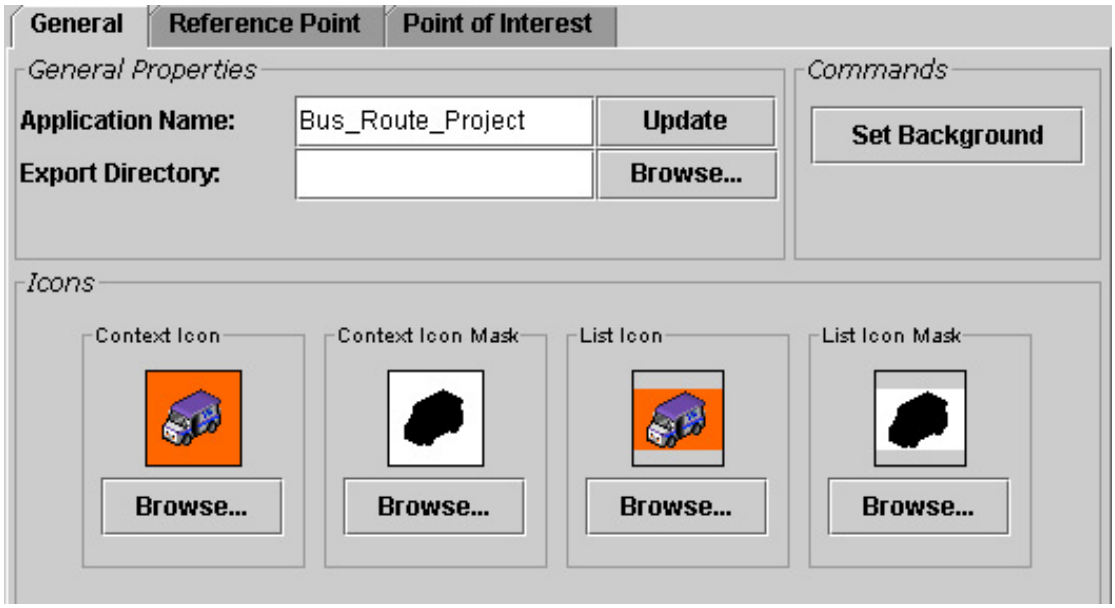


Figure 10.12: General Options Provided in AppGen

AppGen users can set their own application name for both display purposes and source code generation. If the application name provided is Bus_Route_Project, then the class names becomes, for instance, CBus_Route_ProjectAppUi, CBus_Route_ProjectContainer, etc.

AppGen users may also set their own map and icons used in the application. Different image formats are allowed, including JPEG, PNG, GIF, etc.

At the same time, content builders can edit the message displayed in different events, including detection of new reference point, startup, cell ID display format, etc., as shown below:

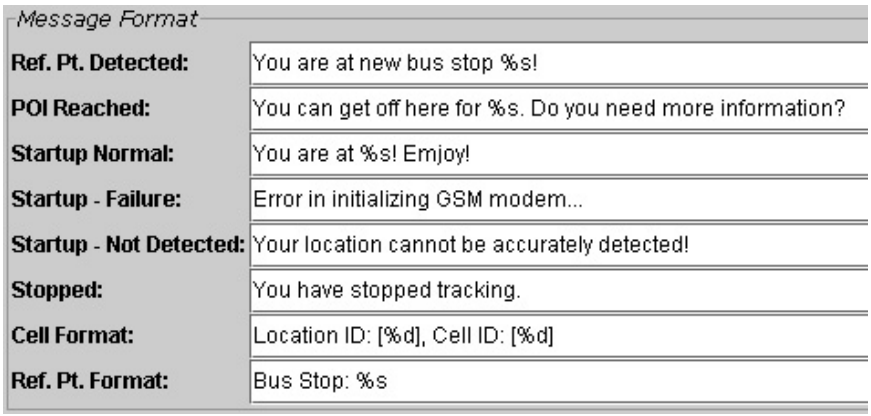


Figure 10.13: Message Editing

Reference points are points taken in Cell Snap and further processed by Cell Analyzer. Usually, they are some fixed points in the target 1D path. The application generated would keep track of these reference points. When user enters a new reference point (or corresponding region), certain actions are taken, such as changing the display at the bottom of the screen or showing a non-modal message box.

Content builders can also associate a pixel point from the map to a reference point such that the screen would change the display (with that map point as centre) when users enter this reference point.

Figure 10.14: Selection of a Point on the Map for a Particular Reference Point

Most importantly, content builders may add certain point of interest (POI) for each reference point (In the starting chapter of middleware series, POI is first introduced). POI is a point or region, of which the

application should be aware, such as Engineering Building in CUHK and University Library. They are not necessarily on the 1D path interested.

By default of AppGen, all reference points are also points of interest. AppGen users can add new points of interest within a reference point. Take MTR case as an example: developers have to map all important constructions around the corresponding MTR station as points of interest.

10.5.2.3 Point of Interest Settings

Most of the configurations are for POI. Content builders are allowed to change the name of POI (most likely the name of particular building or spot) and a description for it, including POI brief description as well as, upcoming news/events of this POI. At the same time, content builder can supply the application with photos for each POI so that application users can have an idea about the appearance before going there.

Figure 10.15: Interface for a Point of Interest

Moreover, developers are able to draw path from the corresponding reference point (e.g. bus stop) to the target POI. For example, the above diagram shows how AppGen users draw a path from a reference point (Central Campus bus stop in CU) to a POI (Engineering Building in CU). This is useful to build a tour guide in the resulting LBS application – telling mobile users to go to a destination from certain well-known points.

10.5.2.4 Source Code Generation

After users have finished their configurations, they can start creating source code line by line by AppGen application. Also, all images are transformed into BMP format for Symbian application.

The following screenshot shows an example application generated without further modification on code. As one can observe, the application can consistently keep track of the change of location ID and cell ID and present user a path from a reference point (indicated by the big dot) to a target destination.

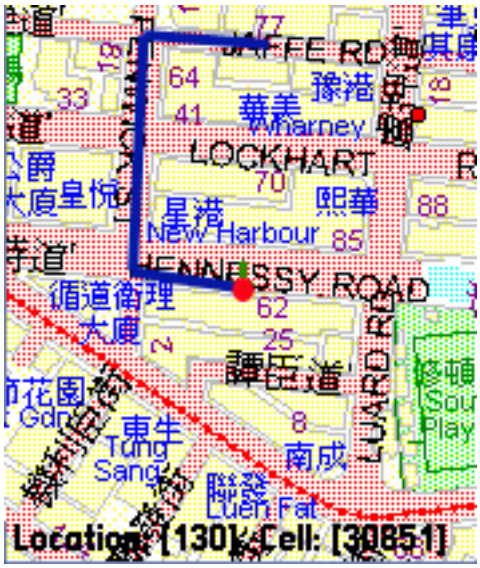


Figure 10.16: A Sample Application Generated

The generated application does already have enough functions to build a general LBS application.

10.5.2.5 Conclusion on AppGen

AppGen provides content builders with user interface to create their applications without writing any code. With control of reference points and points of interest, message formats, POI presentation in detailed text and photo as well as path to destination, generated application would be capable to give enough functions to build a general LBS application. Alternatively, application developers can start building an application from the source code generated.

11. Application Development Paths

With the introduction of our middleware, the original application development path is now modified.

Originally, to develop a location-based application, developers need to under go several fixed steps. Since location-based applications are location-sensitive, developers need to collect location information (location ID and cell ID pairs). After that they have to manually manipulate those location information collected, for example they have to identify, group, match those information. Lastly, based on the processed location information and the specification of the application, developers can start to write their code.

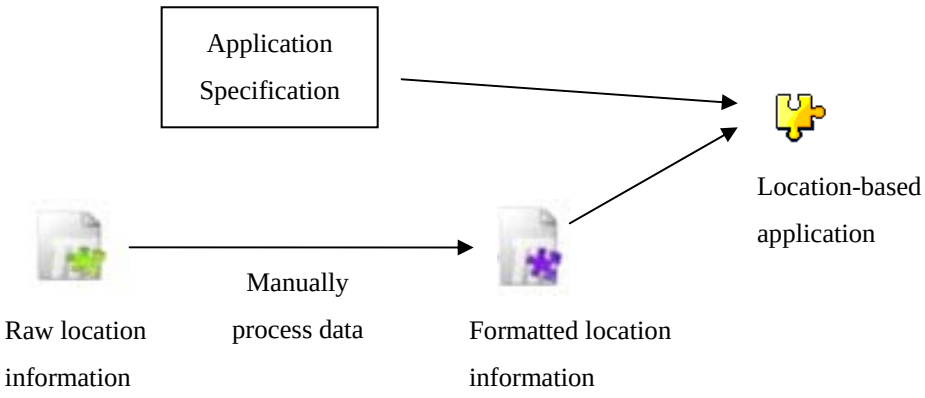


Figure 11.1: Traditional Location-based Application Development Path

With our development kit set, not only is the development process simplified, the number of development paths also increased. The increase in development path does not mean the development process becomes complicated. We provide different development path in a hope to provide flexibility to developers.

11.1 The New Development Paths

The middleware provides developers with totally three different development paths. They are:

1. Full use of the middleware, from data collection, data manipulation, classification to application generation.

2. Use development kit set to collect and process data, but manually write application codes.

3. Manually write the application using the APIs provided.

All the above paths provide different degree of simplicity and flexibility to developers. And we are going to explain their advantages in the following sub sections

11.1.1 Development Approach One

In this approach, we will use Cell Snap, Cell Analyzer, AppGen and the API set. The following figure shows the process:

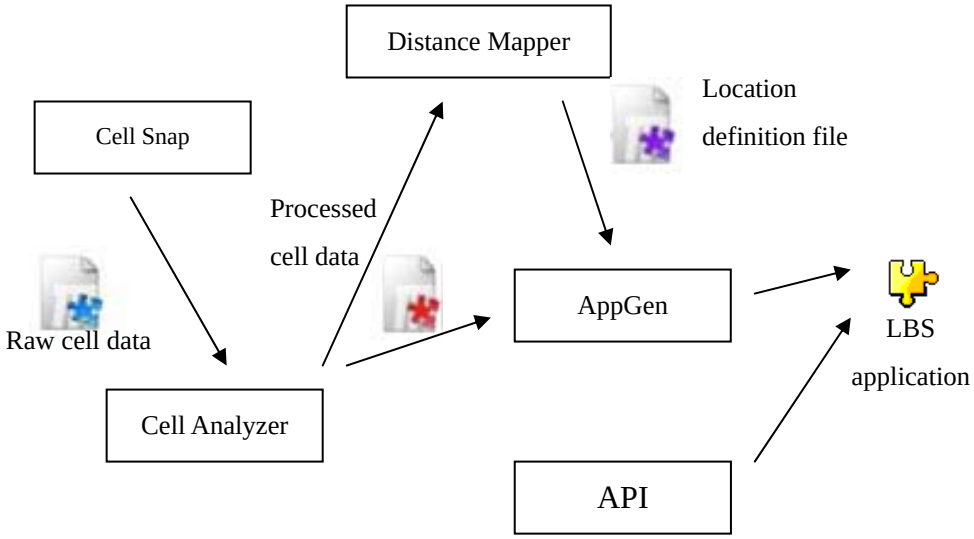


Figure 11.2: Graphical Representation of Development Approach One

Developers use Cell Snap to collect location information, that say, location ID and cell ID pairs. These raw information data are fed into Cell Analyzer for further processing. Developers can correlate

location information collected from different telecommunication companies together, remove duplicated information and add comments. The processed cell data would then be used by Distance Mapper, if the developer wants to provide searching function in the output application. The processed cell data, perhaps together with a location definition file, are supplied to AppGen, which generates location-based application according to developer's parameter selection. The API we provided will be used in generating the output application.

This approach needs the minimum intervention from developers since most of the development processes are automatically accomplished. What the developer needs to do is to collect location information using Cell Snap and select application generation parameter in AppGen. Of course, minimum intervention means least flexibility. Applications generated in this manner have fewer variations.

11.1.2 Development Approach Two

To give more flexibility to developers and more variations to those output applications, approach two is introduced. In this approach, developers do not use AppGen in generating the source code for the application, and only use the tool set to collect and process location information into different output files. Developers then write their applications using these output files. Since AppGen is involved, programs written can be of greater variations and will be more tailor-made to suit different request. However, the development complexity will be slightly higher than using approach one.

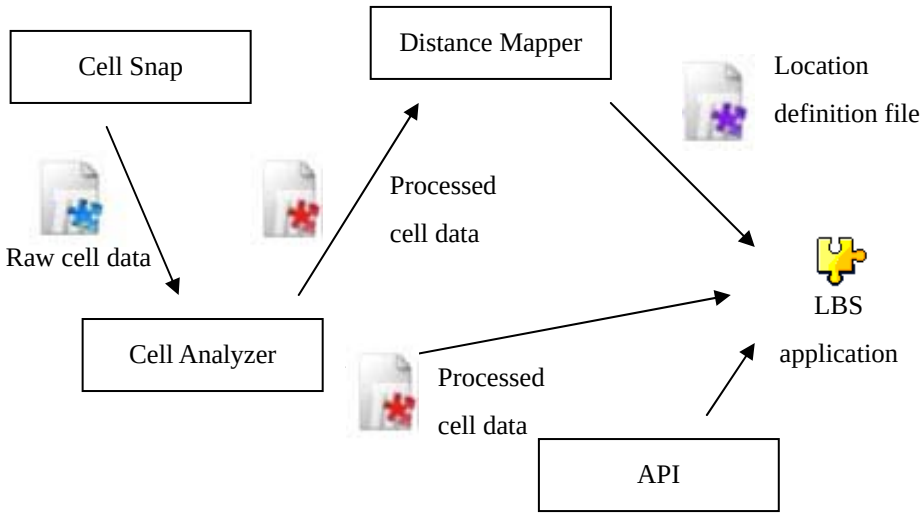


Figure 11.3: Graphical Representation of Development Approach Two

11.1.3 Development Approach Three

This approach give developers the maximum flexibility while providing them with support. Developers may want to create all the data files manually without the need of help from Cell Snap and Cell Analyzer, so that the application written can even be more flexible. In this case, developers only use our APIs to develop location-based applications. They will need to collect and process all location information, and generate data files needed by APIs and their target application.

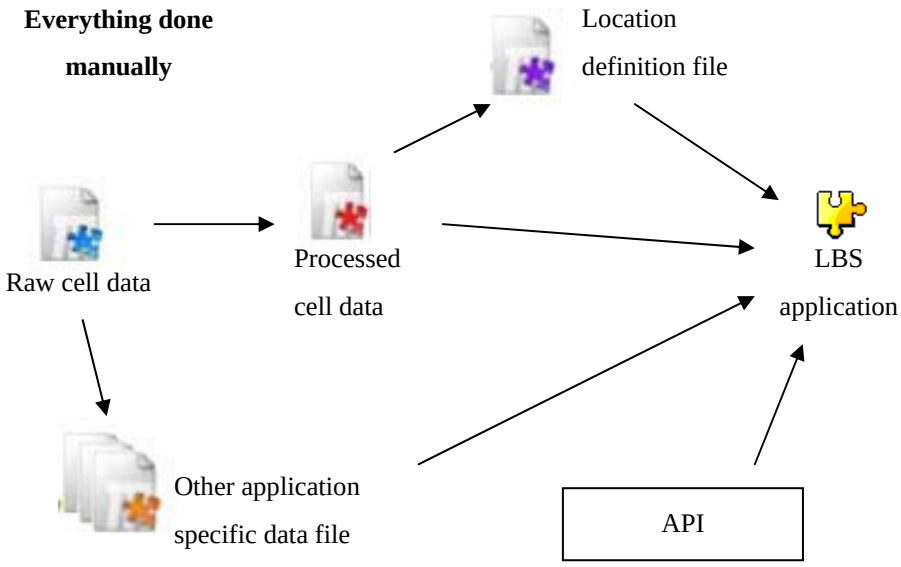


Figure 11.4: Graphical Representation of Development Approach Three

11.2 Classification of Developers

Developers can be classified into three categories according to the path they take in developing location-based applications. The above development paths suit all these three types of developer.

Low-Level Developer:

These kinds of developers would like to work on GSM cell ID and even optimize the underlying mechanism and algorithm to suit their application needs. They may need a simple interface to retrieve and work around with cell information. Approach Three suits them.

General Application Developer:

Rather than purely handling cell information, they may need a set of tools to facilitate cell data collection, analysis and distance measurement on a geometrical map. They can take approach Two

LBS Content Builder:

They would focus on how the service is offered through content enrichment. They only concerns on location changes in order to give relevant response to application users. A simple example is showing a

message if users enter or leave a particular region. As a result, what they concern mostly are the information, in form of text, graphics, media, as well as the subsequent actions allowed. For simplicity, they should take approach One.

To conclude, the middleware is complete in such a way that all these three types of developers can take full advantage of. LBS development process becomes simplified; developers can design location-based application in a more efficient manner.

12. Experiment on Middleware

In this chapter, two sample applications built by the middleware are presented. MTR Traveller remakes show how the middleware can handle the same task in an efficient way. CU Campus Bus Route project demonstrates the how the package can apply in a real situation.

12.1 MTR Remake

Here we would like to recreate the MTR Traveller application (with similar behaviour) from stretch (i.e. from data collection to actual application running on the mobile phone) through the LBS API and SDK provided.

12.1.1 Difference between MTR Traveller and AppGen-Generated LBS Application

There are several differences between these two applications in terms of underlying mechanisms

1. As mentioned in the chapter for MTR Traveller, it depends on the query to Symbian DBMS. However, both of our applications do not have parallel queries to the database. Moreover, Symbian DBMS does not support table joining and some of the SQL statement like UNION. As there is no significant performance and functional gain in using DBMS, the later LBS application simply gets rid of DBMS usage.

2. MTR Traveller has a dedicated algorithm to handle open area problem by classifying cells into transition cells and station cells. As a result, developers have to submit another data file to specify whether a cell is a transition / station cell or not. On the other hand, Cell Analyzer, the cell data processing tools, does have handled such problem by the reference point architecture (i.e. grouping cells under a reference point (MTR station in this case)). The following two figures illustrate the difference in data structures

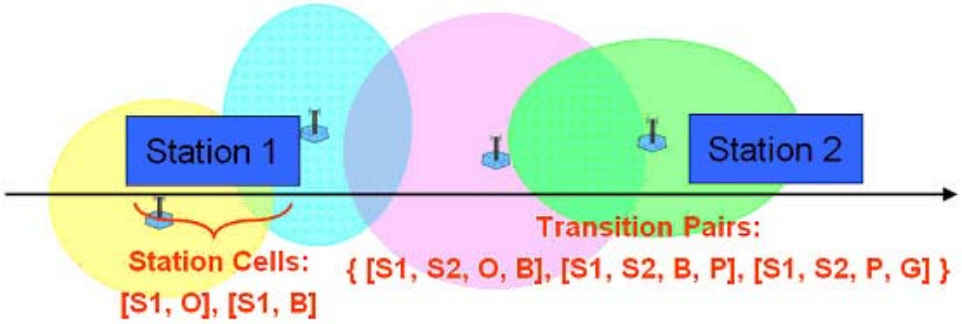


Figure 12.1: How MTRTraveller handles open area stations

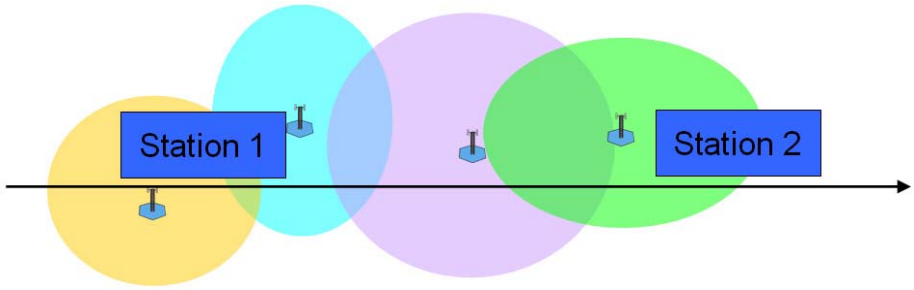


Figure 12.2: How AppGen-Generated application handles open area stations

3. MTR Traveller accesses to GSM modem directly through function calls while the later one is built on the top of LBS API

4. In order to save storage, MTR Traveller chops the MTR / KCR map (in bitmap format) into tiles of stations and rails. However, as the AppGen-created application is designed for general purposes, there is no such storage optimization (i.e. the whole the map is stored rather than constructing the map from tiles)

12.1.2 Data Collection and Processing

With Cell Snap, MTR / KCR data can be simply recorded by taking photos at each station as reference point. Although collectors still has to travel all involved stations, they do not need to manage how cell information changes.

Concerning data processing, MTR Traveller requires developers to:

1. Type in collected data to a text file,
2. Transform cell data into specific format for the program,
3. Identify which cells are station cells and which cells are transition cells,
4. Edit the data by themselves and,
5. Combine data from different network operators manually.

Compared with processing with middleware, developers simply interact with Cell Analyzer user interface and get most of the things done automatically

12.1.3 Application Generation

MTR Traveller was built by writing a Symbian program from stretch. Then, developers have to determine which tiles should be mapped for all stations by finding the pixel coordinates by imaging software.

On the other hand, the later one is created by AppGen. As the concept of point of interest does not exist in MTR Traveller, developers can generate the application by just importing file from Cell Analyzer, entering the project name: MTR Traveller, assigning x-, y-coordinates of the map to each station, and, optionally, importing the icon specific to MTR Traveller. The whole process is less than 10 minutes.

The screenshots of two applications are shown in the following figures:

Figure 12.3: Screenshots for MTR Traveller (Left) and its remake version (Right)

Figure 12.4: Comparison in general MTR map (Left) and arriving a new station (Right)

12.1.4 Comparison between the Two Application

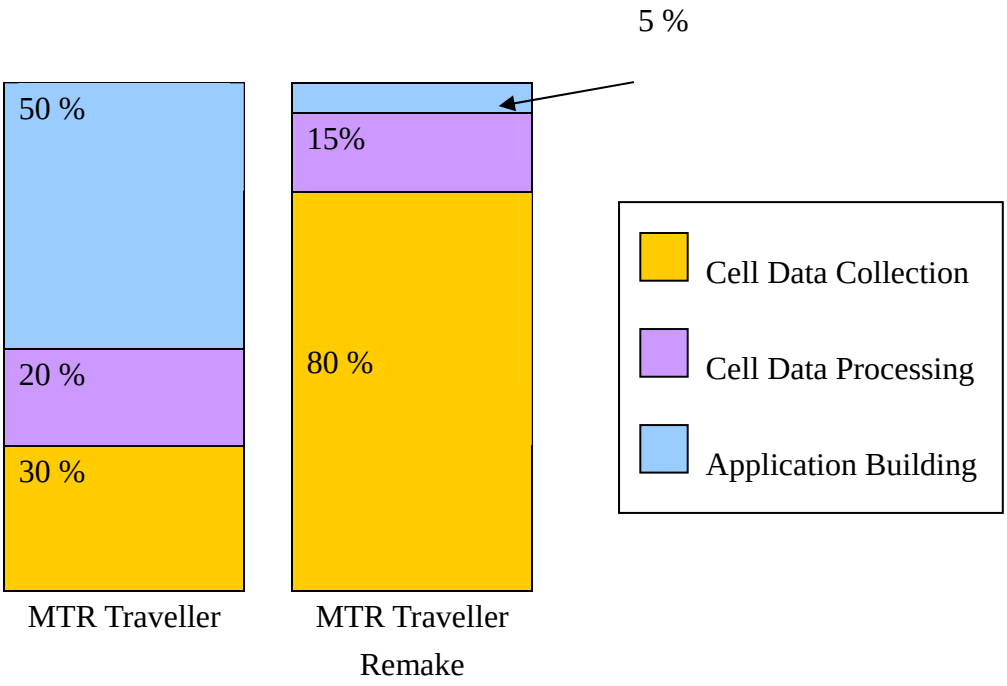
Actually, both of the two applications can have the similar interface and operations. In this section, they are compared with each other in terms of time required for development, package size and application extensibility.

12.1.4.1 Time Required for Development

Both applications contain the data in three of the six MTR routes, namely Tseung Kwan O Line, Kwun Tong Line and Island Line. MTR Traveller required 3-day development with a week for testing and optimization (i.e. 1.5 week in total) for 2 telcos, SmarTone and Peoples.

On the other hand, the remake version was created in less than 8 hours, including data recollection, processing and application building. This shows a significant advantage in time and effort needed for building similar application.

Another interesting point that should be discussed here is the time distribution in each process. The following bars show such factor:



To build MTR Traveller, developers should spend most of time in both data collection (and usually have to collect many times) and application development. However, data collection becomes most significant time contribution in remake version because there is not difference because both cases requires collector to bring the mobile phones to travel around all stations involved. Possible solutions are to put mobile phone devices in MTR trains or to take data from users afterwards when they are using the application.

12.1.4.2 Package Size

The resulting package sizes of two applications can be summarized in the following table:

	MTR Traveller	*MTR Traveller Remake*
Program Binary (i.e. .app file)	76 KB*	60 KB
Package (i.e. .sis file)	42 KB	122 KB

Table 12.1: Comparison of Packet Size

* MTR Traveller binary size is slight larger because it embeds with GSM Status (the program that displays currently registered cell information and signal strength) also. This information is not the concern of the discussion here.

The binary size of two applications can be regarded as similar. However, the package size (i.e. the total size of all files involved, including binary, icons, data files and images) of the remake version is significantly larger than the original one due to the use of full-size bitmap for the MTR route map, compared with bitmap tiles in original version of MTR Traveller.

12.1.5 Application Extensibility

As we know, an application is not designed only for current use. In the future, there may have necessary modifications in order to fit the reality. Consider a real case that a new MTR station, Nam Cheong, came into service on 16th December 2003. In order to capable to handle this new station, the following changes are required for MTR Traveller:

1) collect cell data for this new station,
2) add this entry in the map data file and cell data file manually and,
3) make a station bitmap tile for Nam Cheong station, rearrange the map array value in the resource file as well as recompile the program.

Considering AppGen-generated MTR Traveller, what developers need to do are to:

1) collect cell data for this new station and regenerate a new data file and,
2) replace the new map.

Of course, both applications require developers to recollect the new data. However, MTR Traveller may require recompilation of the program because of the change in resource file, resulting in the situation that the application should be redistributed to users. On the other hand, the later application requires only to replace the bitmap and this can be done through network update without program redistribution. This case middleware provides a nice support when an application extends in terms of time and dependency.

12.1.6 Conclusion

With the help of middleware, developers can reduce the time in producing a LBS application. Meanwhile, the application is more easily to be managed and extended.

12.2 CU Campus Bus Route

This application is created also through the help of middleware. This time, we had collected cell data from 4 network operators, namely SmarTone, Peoples, Sunday and Orange, and it was created at the end of March, after finishing the implementation of all middleware components. The bus route involved is the one from University KCR Station to New Asia College.

Besides the major function of MTR Traveller, this application aims to be more informative so that it can be a complete and interactive CU campus guide. For example, it would show the details and upcoming events (e.g. seminars) within different locations in the campus, photos of each buildings and path to these destinations.

12.2.1 Data Collection and Processing

v We recollected data in CU bus by Cell Snap, instead of applying cell data from experiment at the end of 2003. It is because we merely own campus-wide data for SmarTone and People only and, most importantly, we would like to show the whole process of building a specific application from the very beginning. We have taken all bus stops as reference points. Appendix contains more detailed data files.

The data processing here is completely handled by Cell Analyzer without manual editing (i.e. Cell Analyzer would remove duplicates, intelligently cut reference point, etc.). This process is similar to MTR Traveller, but it requires extra work to map data from different telcos into a single output data file.

12.2.2 Application Generation

AppGen is responsible for application generation for our CU Campus

Bus Route Guide. Additionally, we had added a lot of content using AppGen fron-end. To consider major buildings of the campus, we inserted those buildings (as POI) to corresponding reference points (i.e. campus bus stops). Also, building names, descriptions, events, photos and the paths from a bus stop to destination location are also considered.

The resulting application is shown below. Users have to select the destination before starting tracking (of course they can select destination later on). Once the corresponding bus stop has been reached, related information would be shown up

.

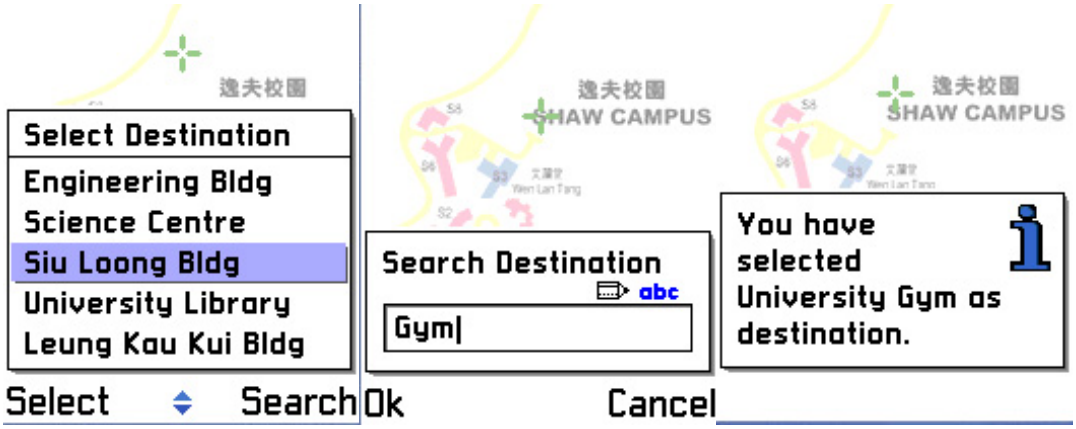


Figure 12.5: Destination Selection (University Gym has been selected)

Figure 12.6: Starting from KCR Station (Left), the mobile phone comes to CC Hostels bus stop eventually (Centre) and shows up information, photos (Right) as well as path from bus stop to University Gym (Centre).

The total development, including cell data collection through traveling with campus bus, processing, building information collection (including photos) and content editing, was done within 1 day which is appreciable.

12.2.3 Potential Problem

As mentioned, we have tested the application with 4 telcos. However, it was found that these four telcos behaves differently. Actually, it represented the same potential problem found in MTR Traveller (imagine when two stations share the same cell, although it does not actually occur from right now).

The most considerable case is Sunday, one of the telcos in Hong Kong, where it had only 3 to 4 cells in the campus, where there were 6 bus stops in the target route.

Figure 12.7: Estimated Cell Coverage of Sunday in CC College

12.3 Trade-off of Using Middleware-Assisted Application

This section concludes the trade-off of using middleware-assisted application and self-built application. Although middleware-assisted application can build an application in a quite development time such that

developers can concentrate on further manual cell data processing and content enrichment, toolkits, especially AppGen, would introduce limitations – LBS application is confined into content providing. For example, when a new LBS game has to be built, there is no way to use AppGen for such purpose. LBS API and Cell Analyzer may somehow restrict developers to have new algorithm for cell data handling and processing. Therefore, there is a trade-off between convenience and flexibility as shown below. The higher the layer is, the more the convenience in LBS application development.

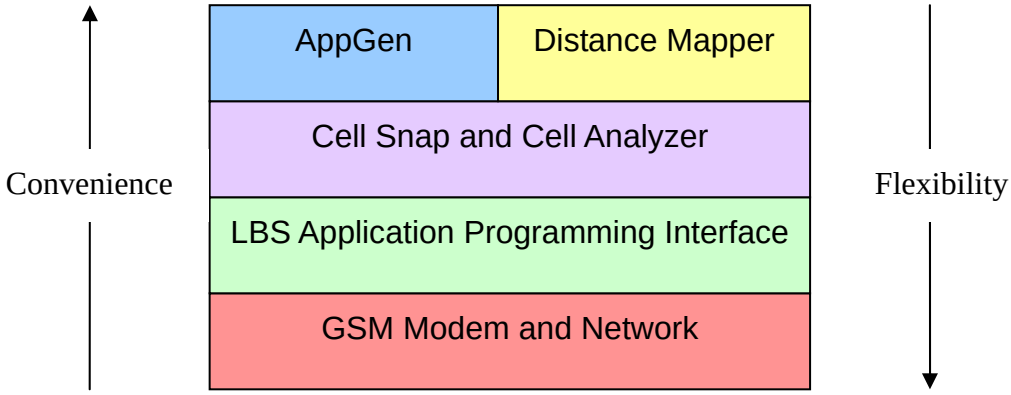


Figure 12.8: Trade-off between Convenience and Flexibility

12.4 Conclusion

From these two applications, MTR Traveller Remake and CU Campus Bus Route Guide, it is found that the middleware assists a lot in automated processing, manual data and content editing and software extensibility – these significantly reduce LBS application development time

13. Conclusion

In this project, we have done quite a number of things. We studied the characteristic and behaviour of the Symbian OS. We also learnt how to program applications used in Symbian OS. Furthermore, we studied some GSM topics and technologies that are useful to our project. They include the system structure and addressing of GSM network, cellular information and cell reselection.

With the basic idea in programming for Symbian OS and GSM technologies, we then investigated current LBS solutions and try to figure out there relative advantages and disadvantages. Then we try to look into the capability of using cell information (location area identifier and cell identifier). The result turned out showed that the accuracy in 2-dimensional space is not accurate without any help from the telco. Then we started to look into the application in 1-dimensional space. The constraints in this space help to eliminate the uncertainties in 2-dimensional space.

Because of the accuracy and certainty in this space, we began to work in the 1-dimensional space. We came up with an idea of providing service and information to tourist in the transportation system. Thus we developed a system called MTR Traveller.

After that, we developed a set of APIs and tool kits used for developing location-based application. This middleware provides LBS developers with three different approaches towards building location-based application. With this middleware, every step in the development process, from location information collection to application source code generation, can be automatic. This makes the development of LBS application more efficient and more reliable.

To be more precise, we have tested the middleware by remaking MTR Traveller and created a new application, CU Campus Bus Route. These two tests shows the middleware works nice.

14. Appendix

14.1 Symbian Location-sensitive SDK Documentation

Class CNetworkInfo

Description:

This class provides basic functions to retrieve the current location data from the mobile phone

void connectL()

This function connects the telephone server through which we can retrieve location data. It must be called before retrieving any location data.

void disconnect()

This disconnects the telephone server.

TBool isConnected()

This function return ETrue if the telephone server is already connected. Otherwise, it returns EFalse.

TInt getCurrentLocationID()

This function retrieves the current location ID and return in form of an integer.

Tint getCurrentCellID()

This function retrieves the current cell ID and return in form of an integer.

Class CLocationListener

Description:

This class provides function that allows users to develop location-sensitive function. By providing a list of desired location IDs, users can specify a function to be executed when the device detects the entrance of any of the cells specified in the list. To use this service, users have to extend this class and implement the function coreFunction().

void start()

Start monitoring cell change event, if an entrance of a specific location specified by the location list is matched, coreFunction() will be invoked.

void stop()
Stop monitoring cell change event.

void connectL()
Connect to telephone server via CNetworkInfo class

void disconnect()
Disconnect from telephone server

void setCheckingInterval(const TInt aInterval)
Set the rate of cell-change event checking. Parameter is time in microsecond. Default checking interval is 1 second

void setLocationList(RArray<TLocation> aLocationList)
Set the location list that is to be checked with the current location IDs. If the location list is empty, coreFunction() will be invoked every time a cell change event is detected

TInt getCurrentLocationID()
Retrieve the current location ID.

TInt getCurrentCellID()
Retrieve the current location ID

void coreFunction()
Users have to implement this function. This function will be invoked if the current location matches any of the location in the location list. This function should take no parameter and return nothing.

void reading(const TDesC& aFileName)
Read in the file specifying a set of reference point. Upon reaching these reference point, coreFunction will be called.

void prepareSelectionList(const TDesC& aFileName)

Read in a file that contains a set of point of interest (POI) together with their relative reference point. The list of POI will be displayed in a list for users to choose.

void setPOI(Tint aIndex) (internal usage)
Set the POI so that when reaching that POI, an action will be taken(prompt message)

Class CProximity

Description:
This class provides functions that allow users to find out the object(s) nearest to the current location.

TBool readIn(TDesC& aFileName)
Read in the distance table specifying the distance of different objects with the current location.

void findNearest(TLocation aCurrentLocation&, RArray<TLocation> aResult)

Given a current location, the nearest object(s) will be inserted into the aResult.
void findSameRange(TLocation aCurrentLocation, TInt aDistance, RArray<TLocation>& aResult) ***(Deprecated)***

Given a current location and a specific distance, it will find out all the object(s) with distance equals to the specified distance. All results

15. Acknowledgement

We would like to thank our final year project supervisor, Professor Michael Lyu, who gave us unlimited support and invaluable advice. He even arranged meeting for us to discuss our project with people from ASTRI. We really want to thank him for his kindness and support.

We would also like to thank Mr. Edward Yau, for he gave a lot of useful ideas and technical help in our project. He enriched our project by providing us with a lot of information we need.

16. References

[1] M. Tasker, Professional Symbian Programming, Wrox, Birmingham, 2002

[2] R. Harrison, Symbian OS C++ for Mobile Phones, Wiley, New York, April 2003

[4] Digia, Programming for the Series 60 Platform and Symbian OS, Wiley, New Your, November 2002

[5] M.J. Jipping, “Telephony,” Symbian OS Communications Programming, Wiley, New York, pp. 350-372, June 2002

[6] “Symbian OS – the Mobile Operating System,” Symbian Ltd., 2003 [Online]. Available: http://www.symbian.com

[7] “Nokia on the Web,” Nokia Ltd., 2003 [Online]. Available: http://www.nokia.com

[8] “Developer Discussion Boards – Symbian,” Nokia Ltd., 2003 [Online]. Available: http://discussion.forum.nokia.com/forum

[9] “Symbian OS Communicators and Smartphones Info Center,” 2002 [Online]. Available: http://my-symbian.com

[10] “All About Symbian,” 2003 [Online]. Available: http://allaboutsymbian.com

[11] J. Eberspacher, H. Vogel and C. Bettstetter, GSM Switching, Services and Protocols, 2nd ed., Wiley, Chichester, pp. 9-92, 2001

[12] M. Mouly and M.B. Pautet, “Current Evolution of the GSM Systems,” IEEE Personal Communications Magazine, October, pp. 9-19, 1995

[13] W.C.Y. Lee, Mobile Cellular Telecommunication Systems,

McGraw-Hill, New York, 1989

[14] “The Website of the GSM Association,” GSM Association, 2003 [Online]. Available: http://www.gsmworld.com
[15] J. Tisal, GSM Cellular Radio Telephony, Wiley, New York, 1997

[16] M. Mouly, M.B. Pautet, The GSM System for Mobile Communications, Palaiseau, France, 1992

[17] S.Y. Willassen, “Positioning a Mobile Station,” March 1998 [Online]. Available: http://www.willassen.no/msl/node6.html

[18] S. Buckingham, “Mobile Positioning – An Introduction,” December 1999 [Online]. Available: http://www.mobilepositioning.com

[19] J.H. Yap, S. Ghaheri-Niri and R. Tafazolli, “Accuracy and Hearability of Mobile Positioning in GSM and CDMA Networks,” 3rd International Conference on 3G Mobile Communication Technologies, pp. 350-354, 2002

[20] J. Costa-Requena, H. Tang and I. Espigares, “Consistent LBS Solution in Next Generations of Mobile Internet,” Parallel and Distributed Systems, Ninth International Conference, Proceedings, pp 637-642, December 2002

[21] “Garmin: What is GPS,” Garmin Ltd., 2003 [Online]. Available: http://www.garmin.com/aboutGPS

[22] “Global Positioning System Overview,” University of Colorado, 2000 [Online]. Available: http://colorado.edu/geograhpy/gcraft/notes/gps/gps.html

[23] K. Nakamura. “The GPS Resource Library,” 2001 [Online]. Available: http://gpsy.com/gpsinfo

[24] “Wireless World Forum,” W2F Ltd., 2003 [Online] Available: http://www.w2forum.com

[25] Bostock and Pollitt, “CPS Location Based Service”, Cambridge Positioning System Ltd., 2003 [Online]. Available: http://www.cursor-system.com

[26] N. Krishnamurthy, “Using SMS to Deliver Location-based Services,” Personal Wireless Communications, 2002 IEEE International Conference, pp. 177-181, December 2002

[27] K. Chadha, “The Global Positioning System: challenges in brining GPS to mainstream consumers,” Solid-State Circuit Conference, February 1998

[28] M. Goller, “Application of GSM in High Speed Trains: Measurements and Simulations,” Radiocommunicataions in Transportation, IEEE Colloquium, May 1995

[29] C. Drane, M. Macnaughtan and C. Scott, “Positioning GSM Telephones,” Communications Magazine, IEEE, vol.50, Issue 4, pp. 44-54,59, April 1998

[30] C.L.C. Wong, M.C. Lee and R.K.W. Chan, “GSM-based Mobile Positioning using WAP,” Wireless Communications and Networking Conference, September 2000

[31] M.P. Wylie-Green, P. Wang, “GSM Mobile Positioning Simulator,” Emerging Technologies Symposium: Broadband, Wireless Internet Access, April 2002

[32] E. Villier, L. Lopes and B. Ludden, “Performance of a Handset-assisted Positioning Method for GSM,” Vehicular Technology Conference, 1999 IEEE 49th, vol. 3, May 1999

[33] M. Silventoinen and T. Rantalainen, “Mobile Station Locating in GSM,” Wireless Communication System Symposium, IEEE, November 1995

[34] M.A. Spriito and M.P. Wylie-Green, “Mobile Stations Location in Future TDMA Mobile Communication Systems,” Vehicular Technology Conference, IEEE VTS 50th, vol. 2, September 1999

[36] S.B. Guthery, Mobile Application Development with SMS and the SIM Toolkit, McGraw-Hill, New York, 2002

[37] L.B. Gwenael, MobileMessaging Technologies and Service: SMS, EMS and MMS, Wiley, Chichester, 2003

[38] J.Y.B. Lin, Wireless and Mobile Network Architectures, Wiley, New York, pp. 1-12, 2001

[39] T. Gunstone, “SMS and Location-based Service in the Travel Industry,” October 2001 [Online]. Available: http://m-travel.com/11009a.shtml

[40] “Location Based Services: Heading in the Right Direction,” January 2001 [Online]. Available: http://www/geoplace.com/ge/2001/0101/0101lbs.asp