Universitat Autònoma de Barcelona

escola d'enginyeria

2169 Estimación de las emisiones de CO_2 vía GPS

Memoria del Proyecto Final de la Carrera de Ingeniería en Informática

Realizado por

Diego González Domínguez

Y dirigido por

Montserrat Meneses Benítez

Bellaterra, junio de 2010

La que firma abajo, Montserrat Meneses Benítez

Profesora de la Escuela de Ingeniería de la UAB,

CERTIFICA:

Que el trabajo al cual corresponde esta memoria ha sido realizado bajo su dirección por Diego González Domínguez

Y para que conste firma la presente.

Firmado: Montserrat Meneses Benítez

Bellaterra, junio de 2010

AGRADECIMIENTOS

En primer lugar a mi profesora de proyecto Montserrat Meneses Benítez por permitirme hacer el proyecto y por animarme a hacerlo por partes, y recordarme que las memorias no se dejan para el final, A toda mi familia por su interés por el proyecto y sus comentarios sobre la aplicación y a los compañeros del SID de informática de L'escola d'enginyeria de la UAB de Bellaterra, Pere Cerdán, Juan Carlos Moure y Juanjo Rodríguez por escucharme y razonar conmigo sobre los diferentes problemas del proyecto cada vez que los explicaba en voz alta.

Índice general

Contenido

Lista de ilustraciones

1. Instrucción

1.1 Presentación

El **Sistema de Posicionamiento Global** fue puesto en órbita por el Departamento de Defensa de los Estados Unidos, la red funciona actualmente con 27 satélites, los cuales dan cobertura al planeta entero. El sistema permite determinar en todo el mundo la posición de un objeto, una persona, un vehículo o una nave, con una precisión hasta de centímetro.

El sistema se basa en tener a la vista al menos 3 de esos 27 satélites para poder hacer una triangulación en el sistema receptor de manera que podemos saber nuestra posición y altitud sobre el planeta tierra.

Depende del número de satélites que tengamos disponibles en cada momento podemos tener una fiabilidad que va de los 2.5 metros a los 15 metros.

1.2 Motivación

En el mundo actual, tan lleno de contaminación y con tantas cumbres sobre el cambio climático, las personas conscientes de tales problemas, buscan maneras de tratarlos y mejorar su vida. En concreto, podrían buscar una manera de reducir poco a poco su consumo de CO_2 llevando el cálculo de cuanto gastan.

Por otro lado hay muchas personas a las que si los problemas no les afectan directamente no reaccionan por lo tanto hay que darles datos o demostrarles que por poco que hagan se pueden cambiar las cosas. Así que pongámosles en sus manos una herramienta que les permita abrir los ojos. El tomar conciencia del daño que ocasionamos nos motivara a actuar de una forma más sensata.

1.3 Objetivos

Nuestro objetivo es el desarrollo de una aplicación que lleve constancia de nuestra huella de CO_2 asociada a nuestro transporte. Lo haremos mediante el GPS, tanto integrados como Bluetooth, los cuales son capaces de dar información sobre la velocidad actual que lleva el vehículo o persona.

Una vez conseguida esa información (la velocidad), se podrá determinar si el ciudadano va en coche, metro o avión.

Por lo tanto, se asociaran las emisiones de CO_2 relacionadas con el transporte y podremos calcular el impacto en el medio ambiente.

1.4 Planificación del proyecto

1 octubre – 30 octubre	búsqueda de información, aplicaciones hechas, datos lenguajes de programación y funcionamiento del GPS
30 octubre – 31 octubre	instalación del entorno de desarrollo
2 noviembre - 31 noviembre	pruebas de programación móvil.
2 diciembre - 11 febrero	test del sistema de GPS integrado en móvil. Aplicación de recogida de datos.
11 febrero – 27 febrero	implementar y desarrollar el sistema de votación que indicara sobre que vehículo nos desplazamos a partir de los datos de velocidad adquiridos.
1 marzo - 15 marzo	desarrollo del sistema que calcula a partir de la velocidad y los datos adquiridos un cálculo del consumo actual de CO_2, en un intervalo de tiempo de desplazamiento.

15 marzo - 1 abril - 15 abril	búsqueda de información, Implementar el sistema de guardado de información en disco para llevar 1 historial del consumo.
15 abril - 30 abril	visualización gráfica de los consumos del historial.
1 noviembre - 15 mayo	realización de la memoria del proyecto.

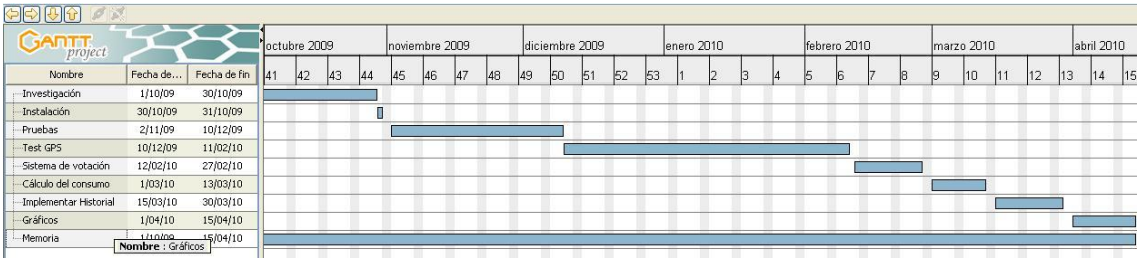

Ilustración 1: Diagrama de Gant

Nombre	Fecha inicio planificada	Fecha final planificada	Fecha inicio real	Fecha final real	Horas dedicadas
Investigación	1-10-09	30-10-09	1-10-09	1-05-2010	60 horas
Instalación	30-10-09	31-10-09	30-10-09	2-11-09	10 horas
Pruebas	2-11-09	10-12-09	3-11-09	10-12-09	30 horas
Test GPS	10-12-09	11-02-09	10-12-09	15-12.-09	20 horas
Sistema votación	12-2-10	27-2-10	12-2-10	15-3-10	100 horas
Cálculo de consumo	1-3-10	13-3-10	15-3-10	30-3-10	150 horas
Implementar Historial	15-3-10	30-3-10	15-3-10	30-3-10	60 horas
Gráficos	1-4-10	15-4-10	1-4-10	15-5-10	70 horas
Viajes de pruebas de aplicación					200 horas
Memoria	1-10-9	15-4-09	1-3-10	30-5-10	130 horas
Reuniones con el director	1-10-9	17-6-10	1-10-9	17-6-10	45 horas
Aplicación de Test para PC	1-5-10	2-5-10	1-5-10	2-5-10	10 horas

Total de horas dedicadas: 640 horas +200 horas en viajes de test +45 horas en reuniones = 830 horas.

1.5 Costes

Recurso	**Remuneración €/hora**
Técnico	25€/h
Director	45€/h
Personal de test	12€/h

Si el proyecto fuera desarrollado por un equipo de personas dentro de una empresa los costes serian los siguientes:

1. Sueldo de un director de proyecto:
   - Reunión de 1,5 hora cada semana: 45 horas
   - Búsqueda de información anterior al proyecto 50 horas

     Total: 95 horas * 45€/h = 4275€.
2. Sueldo de un programador junior:

   Total: 640 + 45 horas *25€/h = 17125€.
3. Licencias usadas por la aplicación:

   Total: 0€.
4. Test de la aplicación sobre vehículos (Gastos de gasolina o billetes de tren, avión, metro):

   Total: 10000€.
5. Sueldo de las personas de test Contratación de 800 horas extra de pruebas.

   Total: 1000 horas *12€/h = 12000€.

Aplicación Completa Total: 43400€.

1.6 Contenido de la memoria

En el tema 1 se presenta la memoria del proyecto sus objetivos planificación y coste del aplicativo.

En el tema 2 se hablara sobre las aplicaciones móviles desarrolladas en java su forma y funcionamiento.

En el tema 3 comentaremos tanto las librerías necesarias como los datos que necesitaremos para hacer los cálculos correspondientes en nuestro proyecto.

En el tema 4 plantearemos los requisitos tanto funcionales como no funcionales del proyecto.

En el tema 5 plantearemos como se llegan a obtener los datos necesarios por el proyecto para el cálculo de valores.

En el tema 6 explicaremos el tratamiento dado a los datos para inferir en que vehículo nos estamos desplazando.

En el tema 7 mostraremos de donde sacamos los valores e consumo mostrados por el aplicativo.

En el apartado 8 explicaremos como guardamos las estadísticas y en que formato.

En el tema 9 se resume el tratamiento y dibujo de estadísticas con las clases canvas de java.

En el tema 10 mostraremos el resultado de la aplicación desde un punto de vista grafico.

En el tema 11 se evalúa el rendimiento de la aplicación comentando entre ellos las razones de esos resultados contra los datos obtenidos por las pruebas.

En el tema 12 comentaremos las conclusiones y resultados expuestos por la versión final.

En el tema 13 expondré mi valoración personal del proyecto.

En el tema 14 mencionare futuras líneas de desarrollo de cara a expandir el proyecto.

Anexo A incluye tanto un manual de instalación, como de funcionamiento formatos de ficheros de salida y la aplicación.

Anexo B menciona las diferentes versiones de desarrollo y sus problemas punto a punto.

Anexo C explica el uso de la aplicación exportada al java de sobremesa para emular recorridos a partir de ficheros de Log de recorridos reales.

Anexo D menciona una comparación de los resultados de las versiones anteriores con la última.

Anexo E menciona las herramientas usadas en el desarrollo y el hardware de test.

2. Aplicaciones de Móviles Java

El primer punto a tener en cuenta es el dispositivo, por lo tanto tenemos que observar como lo programamos o en que lenguaje de programación, también que este sea de amplia aceptación en el mercado actual de dispositivos móviles. La elección hecha, Lenguaje de programación JAVA ME (Mobile Edition), se basa en su amplia aceptación sobre el mercado actual (Marejka, 2008).

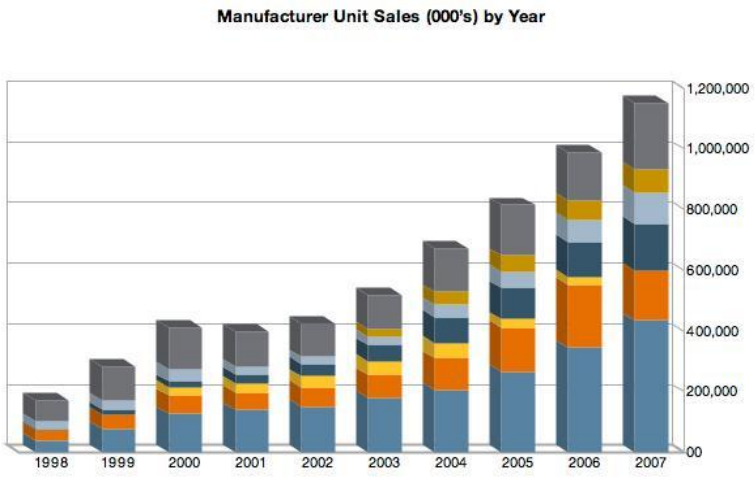

Ilustración 2: Gráfico de las ventas de Java ME en miles de unidades (Marejka, 2008)

En el grafo podemos observar el número de dispositivos móviles vendidos en miles de unidades. En el año 2007, las vendas excedieron de 1 billón de unidades, más de 2.7 millones de móviles vendidos cada día.

Dentro de JavaME tenemos una librería concreta desarrollada para tratar con GPS ya sea este interno al dispositivo o externo Bluetooth. El nombre de la librería es JSR 179 Java Location API. Por lo tanto vemos que java esta extensamente extendido en el móvil hoy día, la mayoría de las aplicaciones de las tiendas han sido diseñadas a través de java por su gran portabilidad, por lo tanto nos centraremos en su estudio para la posterior utilización dentro del proyecto.

2.1 Midlet

Un Midlet es la clase que heredan todas las aplicaciones hechas en java para dispositivos embebidos, más específicamente para la maquina virtual de Java MicroEdition (Java ME). Generalmente son juegos y aplicaciones que funcionan sobre teléfonos móviles. Estando desarrollada bajo la especificación MIDP (Marejka, 2005).

La clase Midlet se hereda en el Core de nuestra aplicación obteniendo sus métodos.

```
public class Application extends MIDlet {
   public Application() { } // constructor de la aplicación

   // Called when the MIDlet is created or re-started
   public void startApp() { } // método llamado cuando la aplicación
se crea

   // Called to pause the MIDlet
   public void pauseApp() { } // que hacer cuando la aplicación esta
pasada

   // Called to terminate the MIDlet
   public void destroyApp(boolean unconditional) { }
// Destructor de la aplicación

}
```

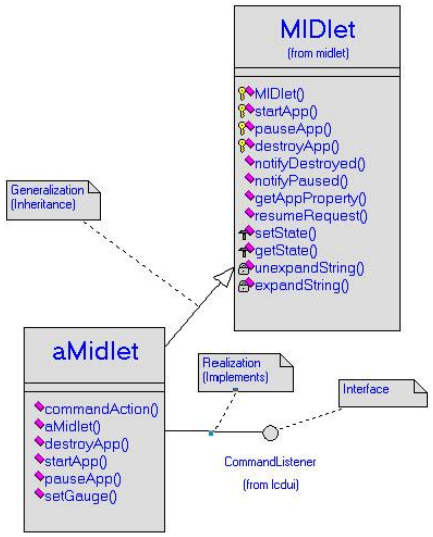

Ilustración 3: Estructura de un MIDlet (IBM, 2003)

2.2 Formularios y Lista

Lo primero que deseamos en cualquier aplicación que se usa cara al público es que sea deseable y tenga usabilidad sencilla, por lo tanto la interfaz de usuario tiene que ser fácil de usar y útil. En concreto usaremos las clases Display, List y Form. La clase Display tiene una función setCurrent(Displayable x), la cual acepta elementos de clase List o Form que usaremos como menús para navegar.

En el menú List Donde podemos poner una serie de opciones por los que movernos con el cursor los cuales al seleccionar uno nos envía a otro menú List o Form.

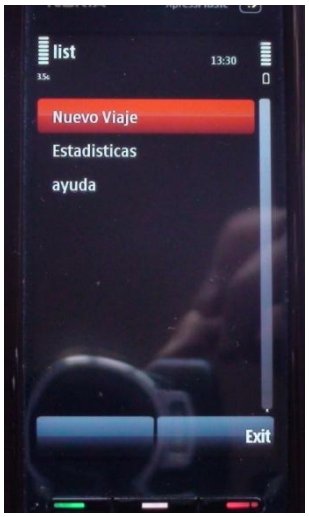

Ilustración 4: Menú List

En el menú Form tenemos formularios en los que mostrar los datos, estos a su vez compuestos de elementos básicos como campos de texto (textField).

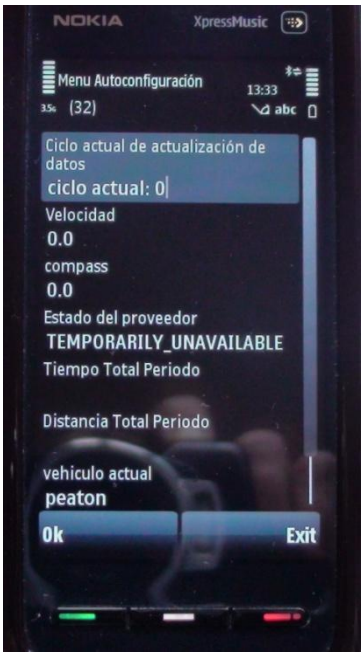

Ilustración 5: Menú Form

Para navegar entre los distintos formularios se necesita respuesta a las entradas del usuario por lo tanto tenemos que implementar la interfaz **CommandListener** Interfaz que nos provee del método **public void commandAction(Command command, Displayable displayable)** mediante ese método podemos usar los métodos descritos a continuación dentro de elementos de formulario o listas de cara a que tengan esa interacción con el usuario.

El método addCommand se usa para añadir botones, que el usuario pueda usar en la interfaz y luego mediante setCommandListener podemos hacer que esa interfaz quede esperando que los botones sean presionados para responder a ellos con la acción pertinente.

3. El procesamiento de datos de GPS

El GPS es el sistema de posicionamiento planetario puesto en órbita por estados unidos de América, se permite su uso público por parte de los civiles, solo se necesita un receptor capaz de obtener los datos enviados por los satélites (NATIONAL MARINE ELECTRONICS ASSOCIATION).

En los sistemas GPS los datos que envían los satélites son Strings de Texto en formato NMEA, el sistema es una especificación eléctrica y de datos para la comunicación entre aparatos eléctricos de la marina, entre ellos el GPS.

Sus parámetros son los siguientes.

Typical bit rate	4800
Data bits	8
Parity	no
Stop bits	1
Handshake	no

Ejemplo de los datos obtenidos por un GPS:

GGA (marimsys)

$GPGGA,hhmmss.ss,llll.ll,a,yyyyy.yy,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx*hh

GGA = Datos del Fijo del Sistema Global de Posicionamiento

1. = UTC de Posición
2. = Latitud
3. = N o S
4. = Longitud
5. = E u O
6. = Indicador de la Calidad de GPS (0=no Válido; 1=Fijo de GPS; 2=Fijo de GPS dif.)
7. = Número de Satélites en uso [aquellos que no se ven]

8. = Dilución Horizontal de la Posición
9. = Altitud de la Antena Sobre/Bajo Nivel del Mar Intermedio (geoide)
10. = Metros (Unidad de la altura de la antena)
11. = Separación Geoidal (Dif. entre elipsoide terrestre WGS-84 y nivel del mar intermedio. -=el geoide está bajo el elipsoide WGS-84)
12. = Metros (Unidad de la separación geoidal)
13. = Intervalo en Segundos desde la última actualización de una Estación de Referencia dif.
14. = Estación de Referencia ID# dif.
15. = Suma de Verificación

Como podemos observar para extraer los datos de estas sentencias se necesitaría construir un parser para adquirir cada valor por separado, con la tecnología actual como veremos más adelante ya disponemos de una librería que de forma optima nos da esa separación además de mas funcionalidades.

3.1 Datos del GPS

3.1.1 Datos de localización

Los datos de localización que vamos a necesitar en nuestra aplicación para poder realizar los cálculos en nuestra aplicación en particular son la latitud, longitud, velocidad, altura, tiempo y brújula.

3.1.2 Datos de Longitud y latitud

Mediante los datos proporcionados por el GPS de latitud y longitud, podemos ser capaces de saber en todo momento en que punto de la superficie terrestre nos encontramos, si esto por ejemplo es comparado con otro punto, podemos calcular la distancia en metros entre esos dos puntos mediante triangulación.

3.1.3 Velocidad

La velocidad nos puede servir para reducir los cálculos, la podríamos usar para multiplicarla por un espacio de tiempo relativamente corto y sacar los metros avanzados por el vehículo.

3.1.4 Datos de altura

Se obtiene la altitud respecto al nivel del mar mediante el análisis de las sentencias nemea. Altitud de la Antena Sobre/Bajo Nivel del Mar Intermedio (geoide), su utilidad es la de ayudarnos a distinguir aviación.

3.1.5 Datos de Tiempo

Las sentencias NMEA incluyen un TimeStamp con el momento exacto en que fueron producidas. Intervalo en Segundos desde la última actualización de una Estación de Referencia dif. Mediante estos datos se puede hacer triangulación entre los diferentes satélites para obtener los datos anteriores.

3.1.6 Brújula

Los datos de brújula siempre son de utilidad ya sea que seamos vehículo como montañista, en nuestra aplicación se centraran en ayudarnos a discernir en que vehículo nos desplazamos, sea coche o trenes por ejemplo. Dado las cantidades diferentes de cambios de dirección existentes entre los distintos modos de locomoción.

3.2 Procesamiento de datos y librerías

3.2.1 Librería

Java Micro Edition tiene una librería llamada Location API (JSR 179 , 2010), esta librería se encarga de buscar receptores GPS al alcance de nuestro dispositivo en función de unos criterios dados, en nuestro caso en los criterios requeriremos que nuestro receptor sea capaz de dar además de latitud y longitud, altitud, velocidad, brújula y coste cero (no nos cobre por triangulación con torres de teléfono).

Nokia nos proporciona una aplicación de pruebas con ejemplos sobre la programación de la librería (Nokia Tourist Route, 2006).

3.2.2 Procesado de los datos

Mediante el uso de la librería de localización obtenemos los valores anteriores. Estos serán usados dentro de nuestra aplicación junto con votadores específicos, los datos se combinaran de cara a maximizar la información que obtenemos de ellos y conseguir inferir el vehículo en el que nos desplazamos.

4. Requisitos y funcionalidad

4.1 Requisitos funcionales

La aplicación debe disponer de tres partes fundamentales: captura de datos, visión de estadísticas y ayuda, accesibles desde el menú principal.

- La captura de datos los cuales son: latitud, longitud, altura, tiempo, velocidad y brújula. Una vez iniciada la captura se dispondrá de un botón con el cual podremos parar la obtención de datos, pasando a una pantalla en la que se muestra el resumen del recorrido. Hecho esto se volverá al menú principal de la aplicación.
- La visión de las estadísticas, tanto kilómetros recorridos, como gramos de CO_2 de cada vehículo, dispondrá tanto de visión mediante gráficos de los valores mes a mes cosa que siempre hace más entendible el peso de cada valor, además también dispondrá de visión los valores numéricos en caso que queramos saber exactamente el consumo de cada medio o la distancia recorrida.
- Por último en la ayuda habrá un resumen del funcionamiento de la aplicación en modo texto para saber las opciones que disponemos dentro del menú de viaje o de estadísticas.

4.2 Requisitos no funcionales

La aplicación debe estar diseñada para dispositivos móviles que dispongan de java. En concreto dispositivos con las librerías java jsr75 (escritura de ficheros), jsr82 (control de bluetooth) y jsr179 (API de localización geográfica) o su equivalente más moderno. También debe ser una aplicación que sea de fácil distribución y user friendly, Además para la elaboración del proyecto no hay presupuestos por lo tanto la aplicación debe ser libre en costes de licencia.

5. Obtención de datos

Para obtener los datos tenemos que mirar el API de localización por satélite para java. Dentro del API tenemos la clase Location, esta tiene los métodos expuestos por la figura siguiente.

Method Summary	
AddressInfo	**getAddressInfo**() Returns the AddressInfo associated with this Location object.
float	**getCourse**() Returns the terminal's course made good in degrees relative to true north.
java.lang.String	**getExtraInfo**(java.lang.String mimetype) Returns extra information about the location.
int	**getLocationMethod**() Returns information about the location method used.
QualifiedCoordinates	**getQualifiedCoordinates**() Returns the coordinates of this location and their accuracy.
float	**getSpeed**() Returns the terminal's current ground speed in meters per second (m/s) at the time of measurement.
long	**getTimestamp**() Returns the time stamp at which the data was collected.
boolean	**isValid**() Returns whether this Location instance represents a valid location with coordinates or an invalid one where all the data, especially the latitude and longitude coordinates, may not be present.

Ilustración 6: Location API (Class Location, 2003)

Como se puede observar entre los métodos anteriores aun no podemos conseguir la longitud o latitud siquiera la altitud para ello se necesita declarar la clase QualifiedCoordinates, esta clase como se ve en la figura siguiente ya tiene métodos mediante los cuales obtenemos los tres valores antes mencionados.

Method Summary	
float	**azimuthTo**(Coordinates to) Calculates the azimuth between the two points according to the ellipsoid model of WGS84.
static java.lang.String	**convert**(double coordinate, int outputType) Converts a double representation of a coordinate with decimal degrees into a string representation.
static double	**convert**(java.lang.String coordinate) Converts a String representation of a coordinate into the double representation as used in this API.
float	**distance**(Coordinates to) Calculates the geodetic distance between the two points according to the ellipsoid model of WGS84.
float	**getAltitude**() Returns the altitude component of this coordinate.
double	**getLatitude**() Returns the latitude component of this coordinate.
double	**getLongitude**()

	Returns the longitude component of this coordinate.
void	**setAltitude**(float altitude) Sets the geodetic altitude for this point.
void	**setLatitude**(double latitude) Sets the geodetic latitude for this point.
void	**setLongitude**(double longitude) Sets the geodetic longitude for this point.

Ilustración 7: Coordinates class (Nokia Class QualifiedCoordinates, 2010)

6. Tratamiento de los datos

Recordemos brevemente nuestro objetivo, un usuario quiere que tras encender su dispositivo GPS y la aplicación que estamos desarrollando capturar los datos de su viaje de una manera que él no tenga que realizar ninguna acción hasta que su viaje acabe, en ese momento el usara el botón de parada de captura de datos. Y se le mostrara un resumen del viaje (km y consumo de CO_2). Por otro lado se tiene que disponer de funcionalidad para ver las estadísticas anteriores mes a mes.

Ahora con esa idea montamos un Diagrama simplificado de la aplicación.

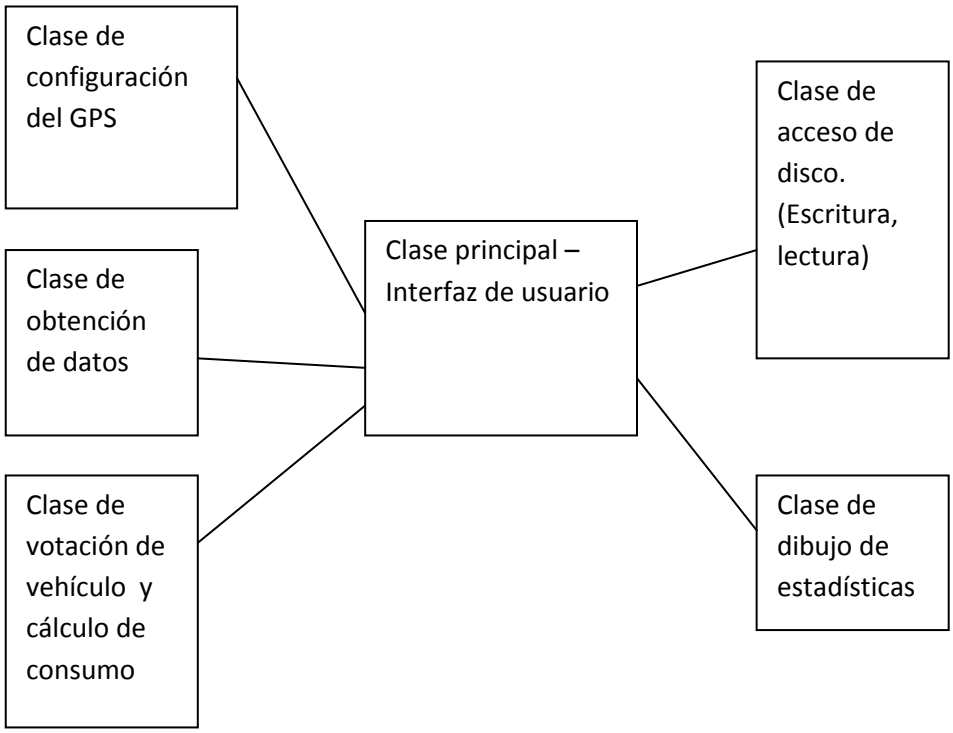

Ilustración 8: Diagrama básico de aplicación

Ahora que tenemos una idea básica de lo que queremos implementar empezaremos con la presentación de los diferentes métodos y sus premisas que llevaron a la

implementación de cara a test, pero primero de todo pongamos presente en nuestra mente unas pocas ideas sobre cómo diferenciar vehículos.

6.1 Sistema de Votadores

La parte del programa más importante dependerá del sistema de votadores, se nos especifica que la aplicación solo depende del usuario para encender y apagar el programa por lo tanto es la aplicación la que debe inferir en que vehículo nos estamos desplazando, para ello tenemos que plantearnos de que valores disponemos y que combinación de ellos nos puede dar más información sobre el vehículo en el que nos desplazamos.

Los valores de los que disponemos como vimos en los datos proporcionados por el GPS son, Latitud, Longitud, Altitud, Velocidad, Cobertura o no cobertura, y curso o brújula.

6.1.1 Votador de velocidad

El más simple de los votadores que se pueden implementar y el primero en que todo el mundo pensaría es repartir los vehículos por rangos de velocidad. Por ejemplo con menos de 10km/h caminar, menos de 18km/h bicicleta, menos de 60km/h coche ciudad menos de 130km/h podría ser tren cercanías o coche autovía, hasta 300km/h tren bala y más de 300km/h aviación.

6.1.2 Votador de Altitud

Manera sencilla de descartar aviones si la altitud es superior a 9km.

6.1.3 Votador de paradas

Cada vez que existe una parada hay posibilidad que se produzca un cambio de vehículo.

6.1.4 Votador de giro

Ejemplificado el problema propuesto por el votador de velocidad en el que tenemos el punto conflictivo de distinción entre coche y trenes cuando su velocidad es mayor a 60km/h, una idea que surgió durante el proceso de desarrollo es la de usar los giros de más de 80 grados, para distinguir entre coche y tren, partiendo de la premisa de que los trenes en un espació corto no giran ángulos tan grandes.

6.2 Aplicación Versión Final: premisas y funcionamiento

Premisas de funcionamiento:

- Las actualizaciones de los datos: latitud, longitud, altitud, tiempo, velocidad y curso son obtenidos por parte del GPS y proporcionados a nuestra aplicación con un rango de tiempo de mínimo un segundo normalmente.
- Los cálculos a evaluar (sea consumo, recorrido, etc….) se harán por rangos de tiempo en su caso 5 ciclos de actualización más o menos de 5 a 6 segundos.
- La distancia recorrida se calculara a través de dos puntos de latitud y longitud los del ciclo 0 y los del último ciclo del rango de tiempo en este caso ciclo 5, a través de una función matemática.
- Cuando en un viaje nos quedamos sin cobertura se guarda la última posición valida (latitud, longitud) y esperamos recuperar señal, una vez recuperada se calcula la distancia recorrida. Que dividido por el tiempo sin cobertura nos da una velocidad media del recorrido.
- Cada vez que llegamos a un punto sin cobertura se hace el cálculo del recorrido anterior, y a partir de ese momento se trata de adivinar de nuevo en que vehículo hemos estado en el momento de sin cobertura basándonos en la velocidad de ese intervalo.
- Después de un intervalo sin señal, el GPS aunque da señal valida es impreciso tiene probabilidad de error de hasta 500 metros por lo tanto se esperan dos ciclos de rangos de tiempo de cálculo antes de empezar a tratar los datos (10 segundos en nuestro caso).
- Un viaje en un vehículo está definido como el tiempo entre que se empieza el recorrido hasta que la velocidad es inferior a 3m/segundo o se pierde la cobertura.
- Antes de decidir en qué vehículo vamos simplemente acumularemos metros recorridos durante un viaje. Una vez el viaje acaba mediante la velocidad media del recorrido y el modulo de control de giro mayor de

80 grados en menos de 3 rangos de tiempo (15 segundos) decidiremos el vehículo actual.

- Premisa de funcionamiento de trenes un tren no gira más de 80 grados en un espacio de tiempo inferior a 15 segundos.
- Cuando la captura de datos concluye se suma el recorrido a las estadísticas del mes actual en disco el fichero tiene formato XML.

Recomendaciones de uso:

- el receptor debería ir en todo momento fijo en el vehículo de transporte usado, de manera que se minimice en ruido indebido debido a la falta de cobertura o movimientos del receptor dentro del vehículo en dirección contraria al mismo.
- Cuando uno se propone a abandonar el vehículo antes de salir corriendo con el receptor si se dispone de tiempo dejar que se estabilice la velocidad al menos durante 5 segundos, de esta manera nos aseguramos que le da tiempo a entender que el vehículo esta detenido.

Factor de decisión para adivinar el vehículo en el que nos trasladamos:

- Mediante la velocidad media de una viaje véase la premisa para la definición de viaje:
    1. Peatón: inferior a 3 metros/segundo
    2. Bicicleta: inferior a 5 metros/segundo 18km/hora
    3. Coche ciudad: inferior a 16.6 metros/segundo , 60km/hora
    4. Coche interurbano autovía: entre 60 y 130km/hora incluyendo algún giro de más de 80 grados entradas salidas autopista, ciudad, curvas, cambios de dirección.
    5. Tren cercanías / Renfe: valores entre 60 y 130km/hora sin giros de 80 grados.
    6. Tren largo recorrido bala, ave: entre 130km/hora y 300km/hora.
    7. Aviación: velocidad media superior a 300km/hora.

Pseudocódigo a alto nivel:

Configuración del GPS y del Listener de datos

IF (Datos del GPS validos)

IF (sin cobertura==true)

Calculo de distancia y velocidad (Tiempo Anterior, Tiempo Actual, latitud Longitud anteriores, Latitud Longitud Actuales)

Vehículo = Votadores (speed, metros, giro 80º)

Calculo de consumo (vehículo, km, CO_2)

ELSE{

Actualizamos valores de captura por pantalla

IF (Ciclo == 5 (maxciclo)) // 5 segundos {

Metros= distancia (latitud longitud anteriores, latitud longitud nuevos)

Speed= metros/tiempo.

Vehículo = Votadores (speed, Metros, giro)

Calculo de consumo (vehículo, km, CO_2)

Ciclo=0

}

Guardamos latitud y longitud validas en memoria

Ciclo++

}

ELSE

IF (Sin cobertura==false) {

Damos por finalizado el recorrido anterior y pasamos a modo sin cobertura

Ciclo = maxciclo

Metros= distancia (latitud longitud anteriores, latitud longitud nuevos)

Speed= metros/tiempo

Vehículo = Votadores (speed, metros, giro)

Calculo de consumo (vehículo, km, CO_2)

Sin cobertura =true

}

ELSE

Esperar recuperación de cobertura

La aplicación contiene un modulo de backup de los recorrido hechos en formato XML los ficheros tienen el siguiente nombre “datos"+hora+"fecha"+dia+mes+año+".xml”, en el cual se escriben los datos de cada conjunto de viajes realizado, entre que empieza la captura de datos hasta que esta concluye. Tiene cuatro diferentes conjuntos de información:

- Tipo Momento: Guarda los datos capturados por el GPS de cada actualización.
- Tipo Viaje: guarda datos posición de inicio y final además de velocidad media de un viaje.
- Tipo no Cobertura: guarda los datos de pos inicio y final de un intercalo sin cobertura además de la distancia recorrida
- Tipo giro: guarda la latitud y longitud del recorrido donde se produce un giro de más de 80º de tal manera que se puede comprobar en mapas estilo Google Maps (Google Maps, 2010).

6.3 Forma final del diagrama de clases

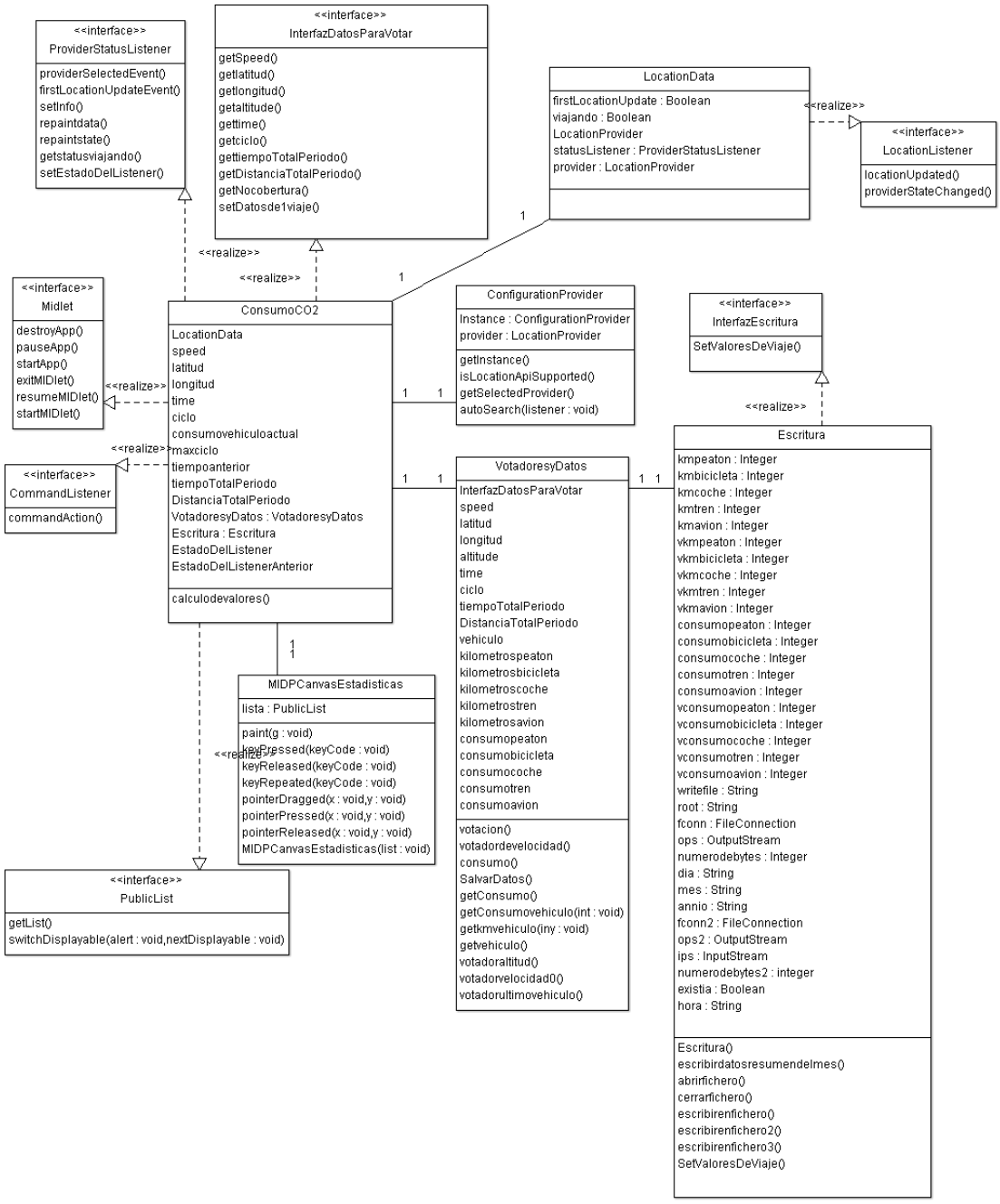

Ilustración 9: Diagrama de clases de la aplicación

Tomando de referencia el diagrama de clases anterior se implementa una clase principal ConsumoCO2, de la que dependerá el interfaz de usuario y su muestra por pantalla de los valores obtenidos por la API de localización.

La declaración de la API de localización estará a cargo de la clase ConfigurationProvider que será un singleton puesto que solo usaremos un dispositivo de captura de datos por lo tanto no necesitamos buscar otro una vez que tenemos nuestro dispositivo instanciado.

Una vez que se encuentre un dispositivo tendremos valores validos de localización, por lo tanto ya podemos instanciar la interfaz LocationListener, de esta manera se llamaran automáticamente a sus métodos **LocationUpdated**() o **providerStateChanged**(), es precisamente gracias a este listener que podremos dejar todo el trabajo de parsear y traducir datos por cada sentencia que nos llega.

El programa estará durmiente y solo ejecuta actualizaciones cada vez que le llegan datos mediante el listener o cuando el usuario decida detener el propio programa.

La clase que se encarga del sistema de actualización de datos obtendrá los métodos de la interfaz providerstatuslistener que entre ellos tendremos dos métodos el primero para repintar los nuevos valores adquiridos y el segundo para repintar el estado del listener en caso que perdamos cobertura de esta manera se lo podremos mostrar al usuario.

7. Consumo de los diferentes vehículos

Para hacer un cálculo de los gramos de CO_2 emitidos se necesita primero buscar información sobre el consumo de los diferentes vehículos que queremos calcular. Para ello nos centraremos en los datos obtenidos por la agencia europea medioambiental (AEMA, 2010). Donde van haciendo recuentos y estadísticas del uso de los diferentes vehículos y sus consumos equivalentes.

Los datos de vehículos serán obtenidos a través del estudio hecho en el siguiente "*Consumo de energía por el transporte en España y tendencias de emisión*" por Pedro José PÉREZ MARTÍNEZ Doctor Ingeniero de Montes, Investigador del Centro de Investigación del Transporte, y Andrés MONZÓN DE CÁCERES, Catedrático de Transportes, Director del Centro de Investigación del Transporte, con la subvención de la agencia española de medio ambiente,

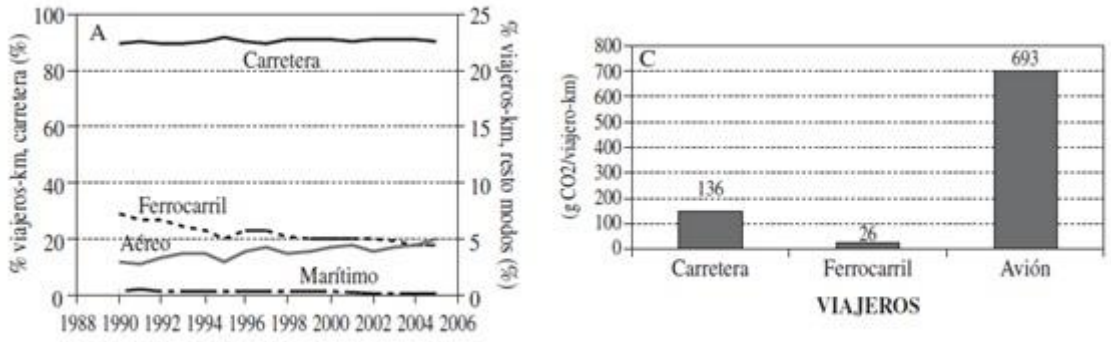

Ilustración 10: Consumo de los diferentes vehículos (Pedro José PÉREZ MARTÍNEZ, 2008)

7.1 Humano

Una persona corriendo emite un CO_2 unos 130g/km podríamos decir que el mismo que un coche por kilometro, está claro que el coche a parte de CO_2 emite otra serie de gases nocivos, realmente como un humano no puede dejar de respirar estableceremos que ir a pie o en bicicleta tienen un coste 0.001 gramos/km meramente informativo.

7.2 Bicicleta

Una persona en bicicleta emite un CO_2 de la misma manera que a pie consideraremos que no hay CO_2 emitido ya que una persona no puede dejar de respirar.

7.3 Coche

Las normas europeas de limitar las emisiones de CO_2 en 2012 hasta los 130 g/km. supondrán unos consumos: 5,49 l/100 km (gasolina) y 4,91 l/100 km (diesel). (TANTAKA, 2007)

Como se puede observar grafícame el consumo actual de media en España según el informe ronda los 136 gramos por kilometro recorrido.

7.4 Tren

Hoy en día la mayoría de los trenes son eléctricos por lo tanto se necesita hacer el cálculo de cuando CO_2 aproximado se usa en el país de origen para producir la electricidad consumida por el tren, otra parte importante del cálculo es la ocupación en proporción a su uso para repartir el coste de funcionamiento por pasajero, de esta manera podremos sumar el peso del individuo dentro del tren y no la totalidad, como se puede observar el ferrocarril suma unos 26 gramos por kilometro recorrido y pasajero.

7.5 Avión

Los aviones son los medios de transporte más rápidos pero a la vez más contaminantes, llegando a sumar un total de 693 gramos por kilometro recorrido.

7.6 Otros

El resto de los datos pueden ser contrastados con el periódico de la vanguardia del día 18 de mayo de 2010 páginas 24 y 25 donde hace una comparación de los diferentes consumos de los transportes.

Medidas de emisión de CO_2 por pasajero y kilometro.

Avión: 405gr/Km recorridos menores a 450km.

Avión: más de 1600km 297gr/km

Coche: recorrido mixto Urbano y carretera 180gr/km

Autobus: 65gr/km

Tren: 60gr/km

Metro: 26gr/km

Fuentes de la vanguardia: Greenpeace, AENA, elaboración propia (La Vanguardia).

8. Guardado y Recuperación de Datos

En Java Micro Edition tiene una API (jsr75) (Mahmoud, 2005) para escribir ficheros llamada FileConnection.

8.1 Tipos de Guardado de datos

La forma más sencilla de guardar datos seria escribir los datos en ficheros XML (w3 schools) de fácil exportación y por lo tanto en un futuro se pueden portar a otros dispositivos como ordenadores y así tener nuestros datos listos para ser importados por una aplicación que lleve cuenta de las estadísticas desde un servidor web por ejemplo. Así mismo los datos en formato XML siempre han sido más fáciles de leer y recorrer gracias al método estándar de escritura que se respeta al escribirlos.

8.2 FileConection

El programa usa estadísticas de los diferentes meses dentro del año para mostrarlas con posterioridad y ver el progreso efectuado sobre el consumo de CO_2 por lo tanto necesitamos un lugar físico donde dejar constancia por escrito de esos valores, para ello escribiremos en disco dentro del sistema de ficheros del dispositivo móvil, de manera que crearemos una conexión a disco. En java para móviles hay una API que nos ayudara en la tarea. Esta API es opcional (JSR-75) es la API que trata con el sistema de ficheros de dos maneras posibles no es necesario que las dos estén implementadas la primera forma de hacerlo es mediante FileConnection.

Podemos observar con el siguiente ejemplo como trata el API a las diferentes unidades de disco del móvil.

Root Value

How to Open a FileConnection

```
CFCard/
FileConnection fc = (FileConnection)
Connector.open("file:///CFCard/");
SDCard/
FileConnection fc = (FileConnection)
Connector.open("file:///SDCard/");
MemoryStick/
FileConnection fc = (FileConnection)
Connector.open("file:///MemoryStick/");
C:/
FileConnection fc = (FileConnection) Connector.open("file:///C:/");
/
FileConnection fc = (FileConnection) Connector.open("file:////");
```

Después de varias pruebas dentro del dispositivo móvil había ciertos problemas con la política de escritura sobre la unidad c: del dispositivo o escribiéndolo de otra manera sobre el root, este problema fue resuelto escribiendo en una unidad SDCard externa donde no hubo problemas de política de seguridad. La unidad externa es tratada con la letra E:/ por el móvil.

El API es capaz de abrir conexiones a disco, estas conexiones se pueden ligar a un InputStream u OutputStream de java, si esto lo juntamos con la abertura del fichero en modo lectura escritura (Read_Write), nos permite trabajar con él como si estuviéramos en plataformas de sobremesa con Windows o Linux.

8.3 Tipo de datos XML

El formato de datos XML es un formato fácil de tratar y recorrer por aplicaciones y navegadores de internet hoy día, por lo tanto es la forma más correcta de salvar los datos y las estadísticas a disco de cara a una futura actualización o post-proceso de los datos adquiridos sus ventajas radican en la gran portabilidad y fácil tratamiento de los mismos frente a una estructura personalizada en fichero binario difícil de exportar con facilidad a un sitio web de terceros.

La problemática de este campo entra en el momento de usar la API de J2me, la cual esta tremendamente subdesarrollada en el java para móvil. Por lo tanto después de investigar por los foros de Nokia, las respuestas recomendadas por los trabajadores era usar una librería gratuita creada por un trabajador de IBM, la librería kXML 2. (kXML) Es de fácil manejo gracias a ella podremos recorrer sin problemas los ficheros XML.

9. Estadísticas

9.1 Navegar por el sistema de ficheros

Las estadísticas están guardadas en la tarjeta SD del dispositivo en formato XML dentro de la carpeta “/data” y con nombres de fichero en formato “mes+año+.xml” la aplicación busca el nombre del fichero del mes actual para mostrar los datos por pantalla.

Formato de ejemplo del fichero de estadísticas de un mes:

```
<?xml version="1.0" encoding="UTF-8"?>
 <kmvsconsumo>
  <kmpeaton>1.3654804345715092</kmpeaton>
  <kmbicicleta>0.7375518679618835</kmbicicleta>
  <kmcoche>6.815748609602451</kmcoche>
  <kmtren>0.0</kmtren>
  <kmavion>0.0</kmavion>
  <consumopeaton>0</consumopeaton>
  <consumobicicleta>0</consumobicicleta>
  <consumococche>926.9418080444339</consumococche>
  <consumotren>0.0</consumotren>
  <consumoavion>0.0</consumoavion>
 </kmvsconsumo>
```

9.2 Lectura de los ficheros

La lectura de los ficheros de datos de disco es necesaria para la recuperación de las estadísticas, como se menciono antes, esta se hace usando el código de la clase de lectura de disco, a través del correspondiente Parser XML.

9.3 Muestreo de los datos

Para mostrar datos hay dos opciones posibles haciéndolo con formularios como veníamos durante toda la aplicación o mostrando los datos de una manera grafica, esto siempre ayuda a ver de una manera rápida y intuitiva la cantidad equivalente de cada valor con respecto a los demás en concreto esto se ve bastante bien con gráficos circulares. Necesitamos una base desde la que partir para tratar con la API de dibujo, después de investigar la clase de dibujo de java se basa en heredar de la clase Canvas (Canvas API).

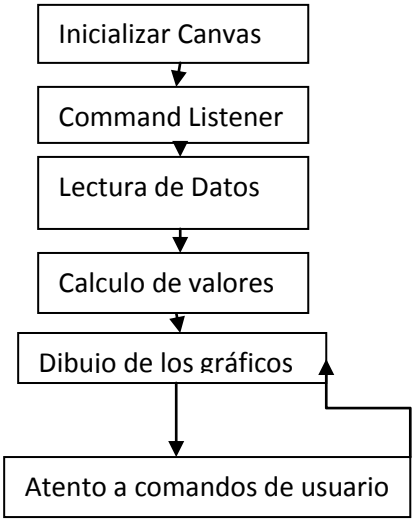

Ilustración 11: Diagrama de flujo Estadísticas

10. Aplicación

Cuando juntamos todos y cada uno de los apartados finales para dar forma a la aplicación obtenemos los resultados que buscábamos al principio una aplicación que consta de 3 módulos básicos.

10.1 Modulo de captura de datos.

El modulo consta de la opción nuevo viaje del menú principal una vez nos encontramos dentro del menú de viaje en curso, solo tenemos una opción que es la de finalizar viaje señalizado por la opción exit. Cuando salimos del modo captura llegamos a la pantalla de resumen del viaje la cual nos muestra los datos obtenidos por el viaje actual kilómetros hechos en cada medio de transporte y su consumo equivalente en gramos de CO_2.

Ilustración 12: Pasos de captura de datos

10.2 Modulo de Muestra de estadísticas

En el modulo de muestra de estadísticas podremos ver las estadísticas de todos los meses que tengamos en disco para el año actual en curso, la aplicación por defecto nos muestra el mes actual, en caso de no tener estadísticas de este mes nos mostrara un mensaje de error.

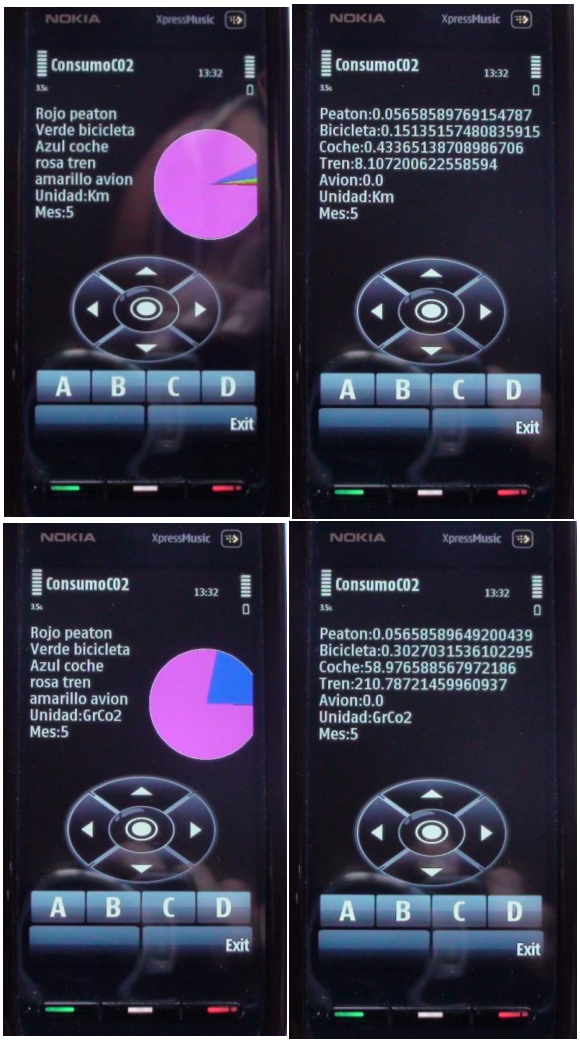

Ilustración 13: Ejemplo de estadísticas

El modulo responde a la interacción del usuario cursor arriba y abajo cambian entre gramos o kilómetros para dibujar el grafico, y los comandos A y C cambian entre dibujar grafico o ver las medidas en unidades numéricas.

10.3 Modulo de Ayuda

El modulo de ayuda consta de una página de texto en el que se resume la funcionalidad de cada uno del los apartados anteriores de forma que alguien con la aplicación instalada sea capaz de aprender las opciones de las que dispone en cada una de las pantallas anteriores.

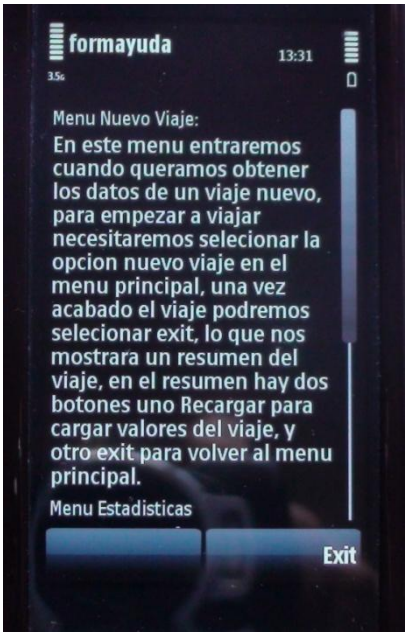

Ilustración 14: Formulario de ayuda

11. Evaluación de los resultados

Para Probar el funcionamiento y el porcentaje de acierto de la aplicación, se han realizado una sucesión de viajes mixtos tanto en coche dentro de ciudad o interurbano, como en trenes de cercanías. Algunos de los viajes anteriores han sido mixtos para comprobar que el sistema es capaz de reconocer el cambio de vehículo sin interacción por parte del usuario. Dentro de cada una de las diferentes fases de diseño se hizo una ronda de veinte viajes durante dos semanas por tal de reconocer exactamente con que peculiaridades del terreno o la falta de cobertura incluía problemas en la recepción o la distinción de los vehículos.

Pongamos por caso el ejemplo del recorrido mixto caminar - tren – caminar - tren - caminar.

Los números siguientes con media de una sucesión de los veinte últimos viajes con la versión final de la aplicación, además de simulación con los datos recuperados de los ficheros de Backup sobre la aplicación portada a PC java de sobremesa.

Vehículo	Kilómetros Reales	Kilómetros Medidos Media	% de acierto. Media
Peatón	1.3	800	61%
Bicicleta	0	300	0%
Coche	0	400	0%
Tren	10.3	9.2	89.3%
Avión	0	0	100%

Se puede observar que aún así los objetivos no fueron conseguidos con un acierto del 100% esto se debe a la falta de precisión en los datos proporcionados por el GPS.

En el caso de ir caminando, el problema viene en el fallo del GPS dependiendo del número de satélites, recordemos que nos da un rango de 2.5 metros de fallo en el mejor de los casos.

- Porque tenemos recorrido de bicicleta? si se supone que se han hecho 0 metros en ella, pues porque durante el viaje se atravesó la facultad de ciencias viaje a través del interior de una facultad, según Google Maps el tramo de facultad atravesado mide 200 metros. La aplicación como hemos visto antes identifica el vehículo por su velocidad media, si hay baja recepción el receptor tiende a casi doblar la velocidad media. Lo que provoca que el tramo sea detectado como bicicleta.
- Porque tenemos recorrido en coche? el receptor de datos nos muestra una velocidad media de coche revisando la simulación y los datos de backup se observa como el receptor se vuelve loco en medio del edificó y nos empieza a dar valores de datos con mala recepción llegando a estar los puntos a 500 metros del sitio real provocando cálculos erróneos tanto en la distancia como en el reconocimiento del vehículo por la velocidad errónea dada por tener que trasladarse a 500 metros en tan poco tiempo.

Pruebas en coche Recorridos urbano – interurbano ejemplo rubí - Park valles

Vehículo	Kilómetros Reales	Kilómetros Medidos Media	% de acierto. Media
Peatón	0		100%
Bicicleta	0	0.5	0%
Coche	11	10.5	95%
Tren	0	0	0%
Avión	0	0	100%

Error provocado por retención en rotonda de estar parado a verse obligado a ir a poca velocidad hasta otra parada.

12. Conclusiones

El programa funciona perfectamente basándose en los objetivos a distinguir dentro de unas limitaciones mencionadas a continuación: Se procurara no estar con el programa capturando datos dentro de edificios en los que haya mucho acristalado, más bien edificios en general salvo que sea bajo tierra o con buen techo, estaciones metro, tren subterráneas, ya que el resto de edificios puede derivar por lo tanto tengamos falsa cobertura, cobertura que influye a los fallos de mala recepción de hasta 500 metros.

El programa actualmente es capaz de reconocer la diferencia en el caso conflictivo de coche interurbano del caso del tren mediante el sistema de control de giros, el único momento en el que no es capaz de distinguirlo es cuando hay un paso de autovía entre dos túneles suficiente largos para perder cobertura, si el vehículo entre ellos no ha realizado ningún cambio de carril se le confundirá con un tren. Ejemplo llegas en taxi a una estación cubierta y coges un tren o metro. Se parecería a ese caso de estudio. No está tratado.

La solución a esta deficiencia seria añadir más modos de recordar el pasado dentro de la aplicación cosa que ramificaría el problema por cada mini caso encontrado.

En el resto de las pruebas realizadas tanto en casco urbano como interurbano se ha detectado que hay suficientes cambios de dirección para que no sea un problema detectar la diferencia entre un coche dentro y fuera de ciudad con un tren sea metro o provincial.

Limitación para detectar coche:

Como se pudo observar en el ejemplo de viaje en coche si no encontramos en un periodo de tiempo extenso casi el 100% de la definición de viaje en el que nos vemos obligados a viajar a una velocidad no normal a dicho vehículo, está claro que el sistema está limitado por los datos que obtiene y no reconocerá ese vehículo como tal ya que mantiene un patrón de movimiento no reconocido por las premisas.

Limitaciones del método sin cobertura:

Cuando nos encontramos en un caso que el vehículo se queda sin cobertura el cálculo entre la pérdida y la recuperación no puede beneficiarse del sistema de brújula, puesto que estamos sin cobertura en ese caso la aplicación seguramente si es a gran velocidad (60-160km/hora) durante el recorrido dirá que nos desplazamos en tren.

La aplicación como tal cumple con su cometido siendo capaz de dar una aproximación bastante fiable de los metros recorridos, por lo tanto podemos obtener el consumo y recorrido siempre que se respete las limitaciones con fiabilidad superior al 90%.

13. Conclusiones Personales

Me siento realmente satisfecho con el proyecto, es cierto que hubo bastantes problemas a encontrar y curiosidades durante el proceso de desarrollo pero cada vez que encuentras o propones una solución a uno y vez que funciona pues te sientes realizado. En cuando al porque decidí hacer este proyecto, las razones son tres básicamente, la primera es mira una aplicación sobre el cambio climático que esta tan de moda por desgracia estamos destruyendo el planeta así que dije voy por ella, la segunda razón fue que la aplicación había que desarrollarla para dispositivo móvil, por raro que parezca durante los estudios de informática en ninguna asignatura tratamos desarrollo para móvil por lo tanto me picaba la curiosidad, llevaba dos años esperando aprender o hacer algo sobre un dispositivo móvil, la tercera no tiene menos interés el titulo del proyecto decía GPS, siempre he sentido muchísima curiosidad sobre su funcionamiento aunque nunca me puse a leer sobre ello en profundidad así que me dije vamos a por ello no quiere decir que al principio cuando no sabía nada de la API, pensaba que sería un horror tratar a bajo nivel con los datos proporcionados por un GPS.

En cuanto a como acaba el proyecto me gustaría haber hecho pruebas en más profundidad o añadir conexión a servidor desde el móvil para subir los datos adquiridos a casa o web, ampliándolo más allá del objetivo inicial, pero las limitaciones en tiempo o problemas adquiridos además durante vacaciones salí de viaje al extranjero al final retrasaron alguna parte que quería llevar a puerto más rápido.

En todo caso ahora con la aplicación probada me siento feliz al ver los resultados de todo ese tiempo invertido.

14. Líneas Futuras

Pero aun podría mejorarse más mediante la inclusión de memoria en el caso de desplazamientos que incluyan largas pérdidas de cobertura, por ejemplo dentro del caso del túnel. Si el tiempo sin cobertura es muy poco, inferior a un minutos lo más probable es que continuemos sobre el mismo vehículo desplazándonos al salir del túnel, esto habría que combinarlo con la velocidad obtenida por el nuevo vehículo puesto que si es diferente ya sea mayor o menor, ósea perteneciente a otro vehículo no deberíamos hacer caso a la memoria, puesto que realmente ha sido un cambio de vehículo, pero en caso contrario el vehículo anterior tiene preferencia.

La segunda línea futura de mejora de la aplicación se basaría en la inclusión de un sistema de base de datos dentro del móvil mediante su inclusión podríamos guardar las estadísticas de los viajes incluso viaje a viaje de esta manera en las estadísticas no solo podríamos tener el valor resumen del mes si no que se podrían crear del día o de la semana.

La tercera manera de ampliar la funcionalidad seria poner una opción de subida de datos a un servidor particular, se podría crear una aplicación web al estilo red social, en el que la gente se registre y tengan sus datos dentro de manera que pueda llevar constancia de los kilómetros recorridos y del consumo de cada viaje. Con opción a mostrar si quiere las coordenadas visitadas y poner fotos del lugar. De manera que encomia a sus conocidos a tomar los transportes públicos que el haya usado hasta el sitio en cuestión compartiendo información sobre un mapa de los lugares, véase (API google maps).

A. Manual de Usuario

A.1 Instalación

Requerimientos:

- Móvil con maquina virtual de java
- API necesarias 75,82 o en su defecto GPS interno y 179.
- Unidad de memoria en el dispositivo con tarjeta SD.

El programa se distribuye como dos ficheros uno ".jar" y otro ".jad" para instalar la aplicación lo único que necesitamos en introducir los dos ficheros dentro de la memoria de nuestro dispositivo móvil con maquina java micro edition. Una vez dentro al ejecutar cualquiera de los dos archivos pedirá confirmación para su instalación, en caso que no deje instalar lo más probable es que necesites a tu dispositivo la instalación de aplicaciones no firmadas.

A.2 Funcionamiento

La aplicación una vez abierta presenta un menú con dos opciones principales, la primera nuevo viaje nos da la opción de empezar el recorrido de un nuevo viaje captando información desde el GPS y guardándola posteriormente a disco junto con el consumo.

La segunda estadísticas nos permitirá ver el consumo de los últimos meses.

A.3 Formato del fichero de salida

Los datos de la aplicación por defecto se guardan en la carpeta data de la unidad extraíble de los dispositivos móviles por defecto la letra de unidad e:/data

A.4 Final de Aplicación

Cuando uno desde el menú de la aplicación desconecta la aplicación mediante el uso del botón salir exit del menú principal regresara al menú del dispositivo móvil desde el cual ejecuto la aplicación.

B. Exposición de las versiones según avanza el desarrollo

B.1 Parte 1

La primera aproximación a nuestra aplicación se baso en las siguientes premisas las cuales fueron la base de test para comenzar a trabajar con el proyecto, algunas de las cuales llegaron hasta la última versión de implementación después de seguir un ajuste a sus parámetros.

Premisas:

La capacidad de cálculo de los dispositivos móviles respecto a la batería es menor que un ordenador de sobremesa. Por lo tanto una buena aproximación seria solo hacer cálculos grandes cada cierto periodo de tiempo, dejando por espacios de tiempo relativos solo la obtención de datos despierta. (Rangos de tiempo entre cálculos).

Cuando uno piensa en rangos de tiempo entre cálculo y cálculo el primer problema se plantea en los momentos de aceleración (arrancada del vehículo), y frenado.

Cuanto tiempo sería adecuado esperar, cuánto tiempo tarda un vehículo coche moto, tren en alcanzar una velocidad adecuada a su tipo de vehículo de manera que en cada votación el consumo se pueda asignar a ese tipo de vehículo.

Un vehículo, ejemplo un coche pasados 20 segundos adquieren normalmente velocidades estables en periodos relativamente cortos de tiempo. De esa manera nos bastara conocer el tiempo inicial y final de un rango de tiempo para que multiplicado por la velocidad nos de él espació recorrido (aproximado).

Pseudocódigo a alto nivel:

Configuración del GPS y del Listener de datos

IF (Datos del GPS validos)

Actualizamos valores de captura por pantalla

IF (Ciclo = rango de tiempo) // 20 segundos

Speed*Tiempodelciclo=metros

Vehículo = Votadores (speed, metros, altura)

Calculo de consumo (vehículo, km, CO_2)

ELSE

No hacer nada.

Problemas:

- Una vez hechos unos cuantos viajes de test se advirtió momentos de pérdida de cobertura en varios puntos, los más claros de mencionar serian estaciones de repostado, túneles de montaña, llegadas a estaciones de tren o cercanías o líneas de metro, cuando no se viaja en coche si uno se sienta por el centro de trenes también se produce más a menudo la perdida de cobertura. Depende de donde se lleva el receptor, como se vio al principio de la memoria una pérdida de señal significa una reducción del número de satélites y un incremento del error recibido llegando a 15m o más lo cual conlleva a tener velocidad por ejemplo cuando no se mueve el receptor.

- A parte la aproximación de tiempo x velocidad = espació con un rango de tiempo elevado incluye mucho error en desplazamientos con gran variación de velocidad (trenes de cercanías).

- El tercer problema viene dado por la premisa de sumar a cada paso el consumo al vehículo cuando un vehículo ejemplo un tren pasa más de 1 minuto sin acelerar a su rango de velocidades en principio propuestas como normales por la premisa y mantiene rato velocidad baja estamos sumando desplazamientos a otros vehículos clasificados como lentos ejemplo coche ciudad o bicicleta. Por lo tanto queda invalidado el rango de 20 segundos como premisa para alcanzar velocidad estable ya que la conducción de vehículos no siempre es ideal, y la segunda razón viene por el error cometido por la aproximación en vehículos con gran variación de velocidad, El metro valles usado para el estudio de trenes, por ejemplo nunca mantiene constante su velocidad por más de 5 segundos siempre tiene variaciones que van desde los 10m/s a los 25m/s.

Propuestas de mejora:

- Implementación de memoria capaz de guardar siempre la ultima latitud y longitud validas en caso de perder cobertura, a la espera de recuperar señal valida. Una vez recuperado el señal valida tenemos cuatros puntos latitud y longitud anteriores a la perdida de cobertura, y latitud longitud validos al recuperar cobertura así podremos hacer un cálculo para obtener los metros entre esos dos puntos de latitud longitud, por lo tanto saber el espació recorrido incluso estando fuera de cobertura un tiempo.

- Reducción a un rango de 5-10 segundos de los periodos de tiempo entre cálculo y cálculo.

- No se votara en que vehículo nos desplazamos hasta que el vehículo se detenga por lo tanto, se usara todo el recorrido a la hora de decidir en qué vehículo nos desplazamos. Recordemos que si el vehículo se desplaza lento lo sumábamos a otro diferente (votador de velocidad), cuando un vehículo puede no cumplir con esa velocidad durante parte de su recorrido.

B.2 Parte 2

Premisas:

Las bases para la versión dos fueron las propuestas de mejora de la primera en resumen, necesitamos añadir un método que aunque no haya cobertura tengamos constancia de los metros avanzados.

Hay que reducir el rango de tiempo para que el error en metros no se acumule y sumar los metros que se van avanzando de manera que una vez este el recorrido completo esos metros pasaran a ser consumo de un solo vehículo. Una persona no cambia de vehículo en marcha.

Pseudocódigo a alto nivel:

Configuración del GPS y del Listener de datos

IF (Datos del GPS validos)

IF (sin cobertura==1)

Calculo de distancia y velocidad (Tiempo Anterior, Tiempo Actual, latitud Longitud anteriores, Latitud Longitud Actuales)

Vehículo = Votadores (speed, metros)

Calculo de consumo (vehículo, km, CO_2)

ELSE {

Actualizamos valores de captura por pantalla

IF (Ciclo == rango de tiempo) // 5 segundos {

Speed*Tiempodelciclo=metros

Vehículo = Votadores (speed, metros, altura)

Calculo de consumo (vehículo, km, CO_2)

Ciclo=0

}

Guardamos latitud y longitud validas en memoria

Ciclo++

}

ELSE

IF (Sin cobertura==0) {

Damos por finalizado el recorrido anterior y pasamos a modo sin cobertura

Ciclo = maxciclo

Speed*Tiempodelciclo=metros

Vehículo = Votadores (speed, metros, altura)

Calculo de consumo (vehículo, km, CO_2)

Sin cobertura =1

}

ELSE

Esperar recuperación de cobertura

Problemas:

- Hubo que obtener una función capaz de calcular el arcoseno de un ángulo puesto que j2me no incluye por defecto esa función (gautier, 2010). Hay una herramienta web muy buena que da la distancia en la referencia siguiente (Calculo de Distancia).

  El cálculo a realizar es el siguiente:

  P = Seno (latitud 1) * Seno (latitud 2) + coseno (latitud 1) * coseno (latitud 2) * coseno (longitud 1 - longitud 2)

  Con ese resultado...
  D = ACOS (P)
  Km = D * 111,194

  Referencia: (larocca)

- Imprecisión en la posición del GPS posición valida improvisada por la API no cierta se revisaron los de varios viajes de test y se dio el caso que a la llegada a una estación de tren edificio cubierto se perdía la cobertura, esto es normal cuando son subterráneas las estaciones pero cuando se salió andando de la estación al exterior recupero la señal valida según la API, pero el punto de latitud longitud coincidía con uno a 500m donde nos hayamos realmente en cambio el punto coincidía con la trayectoria y velocidad anteriores a la pérdida de cobertura. Siguiendo revisando el

código se vio que tardaba más de 15 actualizaciones de datos en reconocer de nuevo una posición realmente estable.

- Mentiras en la velocidad en momentos en los que estas a punto de quedarte sin cobertura (el decremento de señal provoca que la API nos entregue picos de velocidad elevados muy superiores a los llevados realmente) y justo las primeras actualizaciones después de recuperar señal nos entregan el mismo problema, puntas de velocidad superiores al doble de la normal.

- El tercer punto importante es que aun no somos capaces de distinguir entre tren y coche en velocidades entre los 60 y 160km/h

Propuestas de mejora:

Con respecto a los puntos anteriores nos entontáramos con una limitación técnica de la API y el receptor cuando este se encuentra con mala señal. O recién se despierta de recuperando cobertura.

Ideas para recudir el impacto de los valores incorrectos cuando el API recién recupera señal. Esperar las actualizaciones equivalentes a 1 ciclo antes de hacer el cálculo de distancia. Sin cobertura.

Propuestas para la distinción entre coche y tren, se basan en cómo se vio en la parte de votadores en intentar distinguir usando giros de 90 grados en periodos de 15 segundos.

B.3 Parte 3

Premisas:

Un tren no gira más de 90 o más grados en menos de 15 segundos, limitación del giro si lo hacen es a una velocidad tan lenta que no tardaran menos de 15 segundos.

Reduciendo un rango de tiempo antes de volver a calcular valores después de estar sin cobertura se pretende paliar el error y desconcierto sufridos por el receptor justo después de recuperar cobertura.

Problemas:

- Sistema de adquisición de datos de curva por brújula inestable durante los primeros momentos de recorrido ejemplo cuando las brújulas no son magnéticas deciden hacia dónde miras (brújula) por la trayectoria de una sucesión de puntos de GPS, también hay problemas si nos quedamos sin cobertura valida temporalmente por culpa de el mismo problema que la versión 2 de la aplicación la API proporciona valores dispares con mala señal de recepción. (numero de satélites bajo).

Propuestas de mejora:

Hay dos variantes posibles a la hora de mejorar eso la primera se basaría en no usar la API y en modo manual implementar un modulo qué punto a punto de latitud longitud cree un sistema de dos rectas, a las que proyectar una sobre la otra y calcular el Angulo incremental de giro.

La segunda buscar un receptor de GPS cuya brújula interna cumpla con el requisito de ser magnética.

C. Simulación de recorridos a partir de Logs de recorrido reales

Se ha extraído el núcleo de cálculo de la aplicación para móvil y se ha implementado junto con una clase que simula la recepción de GPS a través de los ficheros de log creados por un recorrido real de tal manera que mediante el cambio de parámetros podemos obtener un resumen de lo que ocurre en el viaje de esta manera hacer un estudio sin tener que realizar infinitos viajes reales físicamente.

Como se menciono con anterioridad los ficheros de log de cada conjunto de viajes, se archivan con formato XML y nombres de forma “datos+hora+fecha+dia+mes+año+.xml” en el que tenemos los datos recibidos del GPS segundo a segundo, de ellos se pueden extraer recorridos por defecto. Cada Log usado junto con el corazón de la aplicación exportado a Java de sobremesa permite correr centrares de pruebas a tiempo real de recorridos reales suministrados por testadores de la aplicación. De esta manera podemos ir haciendo pruebas en la modificación de los parámetros de decisión sean tamaño de ciclo límites de velocidad, numero de grados de curvatura de cambios de dirección de modo que ajustáramos mas a la realidad el resultado de la aplicación.

La aplicación se llama Evaluador y es suministrada con el CD de memoria y resto de documentos.

D. Comparativas

Como menciona el anexo B sobre versiones durante el diseño el desarrollo al principio se baso completamente en la premisa de que la velocidad del último intervalo serbia para calcular el espacio recorrido al multiplicarla por el factor de tiempo entre medida y medida si tomamos solo la velocidad cada rango de tiempo (en caso de la versión final 5 actualizaciones) y lo multiplicamos por el tiempo pasado normalmente 5-6 segundos, obtenemos en la mayoría de los casos una aproximación al espacio recorrido haciendo el mínimo calculo apenas una multiplicación. Pero ello en vehículos con mucha variación de velocidad influía en una pérdida de hasta el 50% de los metros recorridos, persona caminando o trenes cercanías con muchas paradas y medio curvas.

Pese a que de esa manera la carga al dispositivo era mínima se decidió ir a por la certeza de los metros recorridos usando el cálculo de distancias de latitud longitud del ciclo 0 contra la latitud y longitud finales de cada rango de tiempo (5 actualizaciones, 5~6 segundos), gracias a esto aunque se perdió en simplicidad de computo ya que ahora hay que llamar a una función que calcula arco seno, no implementada por desgracia en j2me por defecto. Si que hemos ganado una fiabilidad de los metros recorridos de más del 90% salvo en casos que la señal es baja y el GPS nos da datos imprecisos de localización.

E. Programas de desarrollo usados y dispositivos de test

Sun Java Wireless Toolkit for CLDC.

SDK de desarrollo para móviles genérico de j2me de la Sun Microsystems (Sun Java Wireless Toolkit for CLDC, 2010).

Completo IDE de desarrollo que se mezcla con el SDK de móvil NETBEANS (Netbeans, 2010).

Utilidad de Diseño de diagramas UML Argo (argo, 2010).

Testeado el funcionamiento con un Nokia 5800 Express Music en sus 3 versiones de firmware incluida las más actual 50, usando un GPS externo también de Nokia LD-3W.

Bibliografía

AEMA. (2010). Obtenido de http://www.eea.europa.eu/es

API google maps. (s.f.). Obtenido de http://code.google.com/intl/es-ES/apis/maps/

Calculo de Distancia . (s.f.). Recuperado el 2010, de www.tutiempo.net: http://www.tutiempo.net/p/distancias/calcular_distancias.html

Canvas API. (s.f.). Obtenido de http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Canvas.html

Class Location. (2003). Obtenido de http://www.forum.nokia.com/document/Java_Developers_Library_v2/GUID-4AEC8DAF-DDCC-4A30-B820-23F2BA60EA52/javax/microedition/location/Location.html

gautier, b. (2010). Obtenido de http://www.javafr.com/codes/FONCTION-INVERSE-SINUS-ARCSIN-INVERSE-COSINUS-ARCCOS-J2ME_34312.aspx

Google Maps. (2010). Obtenido de http://maps.google.es/

IBM. (2003). Obtenido de http://www.ibm.com/developerworks/rational/library/805.html

JSR 179 . (2010). Obtenido de Java Location API: http://jcp.org/en/jsr/detail?id=179

JSR-75. (s.f.). Obtenido de http://jcp.org/en/jsr/detail?id=75

kXML. (s.f.). Obtenido de http://kxml.sourceforge.net/kxml2/

larocca, s. (s.f.). Recuperado el 2010, de http://www.tutiempo.net/silvia_larocca/Temas/Consultas3.htm

Mahmoud, Q. (2005). Obtenido de http://developers.sun.com/mobility/apis/articles/fileconnection/

Marejka, R. (2005). Obtenido de http://developers.sun.com/mobility/learn/midp/lifecycle/

Marejka, R. (2008). Obtenido de http://java.sun.com/developer/technicalArticles/javame/mobilemarket/

marimsys. (s.f.). Recuperado el 2010, de http://www.marimsys.com/paginas/nmea_codigo.htm

NATIONAL MARINE ELECTRONICS ASSOCIATION. (s.f.). Recuperado el 2010, de http://www.nmea.org/content/nmea_standards/white_papers.asp

Nokia Class QualifiedCoordinates. (2010). Obtenido de http://www.forum.nokia.com/document/Java_Developers_Library_v2/GUID-4AEC8DAF-DDCC-4A30-B820-23F2BA60EA52/javax/microedition/location/QualifiedCoordinates.html

Nokia Tourist Route. (2006). Obtenido de http://www.forum.nokia.com/info/sw.nokia.com/id/f7e8ad78-7898-4053-ab83-74c147923866/MIDP_Location_API_Example_Tourist_Route_v1_0.zip.html

Pedro José PÉREZ MARTÍNEZ, A. M. (9 de 2 de 2008). *Consumo de energía por el transporte*. Obtenido de http://revistas.ucm.es/cca/11391987/articulos/OBMD0808110127A.PDF

TANTAKA. (2007). Obtenido de http://motorfull.com/2007/03/relacion-entre-consumo-y-emisiones-de-co2

w3 schools. (s.f.). Obtenido de http://www.w3schools.com/xml/default.asp

FIRMADO: DIEGO GONZALEZ DOMINGUEZ

Bellaterra, Mayo de 2010

Resum

El projecte a sigut realitzat a petició de Montserrat Meneses Benítez, per al departament de telecomunicacions e enginyeria de sistemes de l'escola d'enginyeria (Universitat Autònoma de Barcelona), el projecte consisteix en la captura de dades per mitja del sistema GPS, mitjançant aquestes dades tenim que endevinar en quin vehicle ens desplacem per a portar el càlcul del consum del CO_2 dels nostres desplaçaments. El programa ha sigut desenvolupat per ser funcional a sobre de dispositius mòbils que tinguin targeta de memòria externa i Java J2ME , inclou interfície gràfica .

Resumen

El proyecto ha sido realizado a petición de Montserrat Meneses Benítez, para el Departamento de Telecomunicaciones e Ingeniería de Sistemas de la escuela de ingeniería (universidad autónoma de Barcelona), El proyecto consiste en la captura de datos por medio del sistema GPS, mediante estos datos tenemos que adivinar en que vehículo nos desplazamos para llevar un cálculo del consumo de CO_2 de nuestros desplazamientos. El programa ha sido desarrollado para ser funcional sobre dispositivos móviles que tengan tarjeta de memoria externa y Java J2ME, incluye interfaz grafica.

Abstract

This project has been developed by request of Montserrat Meneses Benítez, for the Department of telecommunications and Systems Engineering of the Engineering School (Universidad Autónoma de Barcelona), This project consist in the capture of data using a GPS system, with this data we need to foresee in which vehicle we are traveling on, to calculate the CO_2 of our travels. This program has been build to work under mobile devices that have External memory card and Java J2ME, graphical interface included.