ISSN: 1692-7257 - Volumen 1 - Número 27 - 2016

Revista Colombiana de Tecnologías de Avanzada

Recibido: 12 de septiembre de 2015
Aceptado: 29 de octubre de 2015

APLICACIÓN MÓVIL EN ANDROID PARA EL MONITOREO DE RUTAS DE CAMIONES EN LA CIUDAD DE MÉXICO

ANDROID MOBILE APPLICATION FOR MONITORING TRUCK ROUTES IN THE MEXICO CITY

MSc. Tatiana Mileydy Leal del Río*, PhD. Antonio Gustavo Juárez Gracia.*
PhD. Luz Noé Oliva Moreno.**

Instituto Politécnico Nacional (IPN).
***Centro de Investigación en Ciencia Aplicada y Tecnología Avanzada (CICATA). Unidad Legaria**.
Calzada Legaria No. 694, Miguel Hidalgo, Irrigación, 11500, Ciudad de México,
Tel: +(521) 55 5729 6000, Ext: 67780.
E-mail: mileydy.1125@gmail.com, agjuarez@ipn.mx
****Escuela Superior de Computo (ESCOM).**
Ave. Juan de Dios Bátiz esq. Av. Miguel Othón de Mendizába, Gustavo A. Madero,
Lindavista, CP. 07738, Ciudad de México, Tel: (+521) 55 5729 6000.
E-mail: loliva@ipn.mx

Resumen: Este trabajo describe el desarrollo e implementación de una aplicación móvil en Android, que a través de una comunicación *Bluetooth*, permite obtener, administrar y visualizar, los datos de posicionamiento geográfico de un sistema electrónico instalado en camiones, que realizan su recorrido en la Ciudad de México.

Palabras clave: Aplicación en Android, *bluetooth*, SQLite, listas, mapas.

Abstract: This paper describes the development and implementation of a mobile application for Android, using a *bluetooth* communication to obtain, manage and visualize geographic positioning data of an electronic system installed on trucks travelling within Mexico City.

Keywords: Android application, bluetooth, SQLite database, list, maps.

1. INTRODUCCIÓN

Hoy en día los dispositivos móviles inteligentes son cada vez más potentes, con mayores capacidades en memoria **RAM, SD CARD's** y velocidad en el procesador, funciones de entretenimiento, posicionamiento geográfico, e implementación de diversos protocolos de comunicación (*Yan & Shi, 2013*). Android, que es el sistema operativo de Google de código abierto, actualmente está siendo adoptado por la industria y los usuarios de teléfonos inteligentes, para el desarrollo de aplicaciones móviles de entretenimiento y/o científicas, integración de tecnologías a través de servicios web, conexiones Wi-Fi, comunicaciones de datos por Bluetooth y/o NFC, posicionamiento geográfico, bases de datos, aplicaciones en domótica (*Gung et al, 2011)* y el área automotriz, entre otros. Bluetooth, por ser una tecnología inalámbrica y de corto alcance, se está empleando para la transmisión y recepción de datos, que pueden ir desde aplicaciones interactivas para viviendas inteligentes (*Chun et al.,2011; Yan & Shi, 2013*), salas de chat (*Pan et al., 2012*),

sistemas de bloqueo remoto y seguridad en el hogar (*Jeong et al., 2014; Potts & Sukittanon, 2012*), hasta aplicaciones de posicionamiento geográfico de vehículos y usuarios (*Blanco & Romero, 2010; Cedeño, 2013; Gonzáles et al., 2012*) con Bluetooth o servidores y navegadores web, mediante el empleo de los mapas de Google (*Pacheco, 201*), entre otros. En este trabajo se describe el desarrollo de una aplicación móvil en android, que mediante una comunicación Bluetooth entre el dispositivo móvil y un sistema electrónico de posicionamiento geográfico instalado en un camión, se pueda registrar y almacenar el recorrido realizado por el vehículo en la Ciudad de México, de tal forma que se pueda trazar, monitorear y optimizar la ruta realizada.

2. *BLUETOOTH*

Bluetooth es un sistema de comunicación de corto alcance que opera en la banda **ISM** (*Industrial, Científica y Médica*), y fue desarrollado en el año de 1994 para reemplazar los cables que conectan equipos fijos y portátiles vía RS-232 (*Potts & Sukittanon, 2012*). Esta tecnología inalámbrica se centra en la robustez, el bajo consumo de potencia, la seguridad y el bajo costo de implementación. Cualquier sistema con tecnología Bluetooth, consiste de un receptor y emisor **RF** (*Radio Frecuencia*), un sistema banda base, y un conjunto de protocolos (*IEEE Standars, 2005; Téllez et al., 2008-2009*). La plataforma de Android incluye soporte para las redes Bluetooth que le permiten al dispositivo intercambiar datos de forma inalámbrica con otros dispositivos Bluetooth (*Chung et al, 2011*). El marco de aplicaciones de Android provee acceso a todas las funcionalidades a través de las **APIs** de Bluetooth en Android (*Pan et al, 2012; Potts & Sukittanon, 2012*). Usando las **APIs** de Bluetooth, una aplicación en Android puede realizar lo siguiente (*Bluetooth Connectivity; Android Developers*): buscar dispositivos Bluetooth, solicitar el adaptador local (*BluetoothAdapter*) para emparejar dispositivos Bluetooth, establecer canales **RFCOMM**, conectar a otros dispositivos a través del servicio de descubrimiento, transferir datos desde y hacia otros dispositivos, y manejar multiples conexiones. La aplicación móvil sólo contempla conexiones punto a punto entre dos dispositivos Bluetooth. La descripción de los componentes básicos para el manejo de las **APIs** de *Bluetooth* en Android, son:

- **BluetoothAdapter:** Representa el adaptador Bluetooth local (*Bluetooth radio*).
- **BluetoothDevice:** Representa a un dispositivo Bluetooth remoto.
- **Socket:** Es un método para la conexión entre un programa cliente y un programa servidor en una red, y se define como el punto final en una conexión.
- **BluetoothSocket:** Representa la interface para un socket Bluetooth (*similar a un socket **TCP***). Este es el punto de conexión que permite a una aplicación intercambiar datos con otro dispositivo Bluetooth vía **InputStream** (*flujo entrante de datos*) y **OutputStream** (*flujo saliente de datos*).
- **BluetoothServerSocket:** Representa un socket servidor que está escuchando por peticiones de conexión entrantes (*similar a un serverSocket **TCP***).

3. METODOLOGÍA

La estructura de aplicación móvil en Android, se describe de acuerdo a cada bloque funcional: Bluetooth, Base de Datos SQLite, Lista de datos y Mapa de datos (Fig. 1).

Fig. 1. Estructura de la aplicación en Android.

3.1 Bluetooth

La comunicación Bluetooth en Android, se maneja a través de **API's.** Las funcionalidades de conectar y/o desconectar el adaptador Bluetooth local, buscar, emparejar y mostrar los dispositivos Bluetooth cercanos y emparejados, enviar y recibir datos, se implementan en esta aplicación, de forma similar al proceso descrito por *Pan et al., 2012; Pott & Sukittanon, 2012* y *Chung et al., 2011*. Para usar el Bluetooth en el dispositivo móvil, es necesario declarar en el Android Manifest del proyecto los siguientes permisos:

```
<uses-permission
android:name="android.permission.BLUETOOTH
" />
<uses-permission
android:name="android.permission.BLUETOOTH
_ADMIN" />
```

La aplicación móvil desarrollada, permite que el adaptador Bluetooth local, pueda comportarse como servidor o cliente, dependiendo de si este inicia o acepta una conexión, haciendo que la aplicación sea más robusta.

3.1.1 Conexión como Servidor

Para establecer una conexión Bluetooth, un dispositivo debe operar como servidor manteniendo abierto un **BluetoothServerSocket** que este escuchando por peticiones de conexión entrantes. Cuando la conexión es aceptada, el servidor provee un **BluetoothSocket** para la conexión, y descarta el **BluetoothServerSocket**, hasta que el usuario decida iniciar una nueva conexión con otro dispositivo (*Bluetooth Conectivity*). Los pasos básicos para configurar el dispositivo Bluetooth como servidor son:

- Conseguir un **BluetoothServerSocket** llamando a **listenUsingRfcommWithServiceRecord(String, UUID)**. El **String**, es el nombre del servicio y el código **UUID** (*Identificador Único Universal*) se usa para la conexión con el cliente. Cuando el cliente intenta conectarse con este dispositivo, identifica el servicio a través del código**UUID**.
- Iniciar la escucha de peticiones de conexión entrantes llamando el método **accept()**. Esta es una llamada bloqueante, y retornará con una conexión cuando esta sea aceptada. La conexión es aceptada solamente cuando el dispositivo remoto ha enviado una petición de conexión, acompañada de un código **UUID** que coincida con el único registrado por el **BluetoothServerSocket** que escucha. Cuando la conexión es aceptada, el método **accept()** retorna con un **BluetoothSocket** conectado.
- Cerrar la conexión mediante la llamada al método **close()**. Esto liberará al **BluetoothServerSocket** y todos los recursos, pero no cerrará al **BluetoothSocket** conectado que ha sido retornado por el método **accept()** (*El protocolo **RFCOMM** solamente permite que un cliente sea conectado por un canal a la vez*).

Los métodos, y parte del código que implementa este hilo son:

Método HiloServidor(): Este método se encarga de crear un socket para escuchar y aceptar las peticiones de conexión entrantes.

```
public HiloServidor(){
debug("HiloServidor.new()","Iniciando metodo");
BluetoothServerSocket tmpServerSocket = null;
// Creamos un socket para escuchar las peticiones de conexión
try{
tmpServerSocket = bAdapter.listenUsingRfcommWithServiceRecord(NOMBRE_SEGURO, UUID_SEGURO); }
catch(IOException e) {
Log.e(TAG, "HiloServidor(): Error al abrir el socket servidor",e); }
serverSocket = tmpServerSocket;
}// HiloServidor()
```

Método run(): Este méttodo maneja todas las solicitudes de conexión entrantes, y las conexiones que son aceptadas.

```
public void run(){
debug("HiloServidor.run()", "Iniciando metodo");
BluetoothSocket socket = null;
setName("HiloServidor");
setEstado(ESTADO_ATENDIENDO_PETICIONES);
while(estado != ESTADO_CONECTADO){
try { //Cuando un cliente solicite la conexión se asignara valor al socket..
socket = serverSocket.accept();}
catch(IOException e){
Log.e(TAG, "HiloServidor.run(): Error al aceptar conexiones entrantes",e);
break;}
... }//run()
```

Método cancelarConexion(): Es el método que se encarga de liberar el **BluetoothServerSocket** y todos los recursos que este maneja.

```
public void cancelarConexion(){
debug("HiloServidor.cancelarConexion()",
"Iniciando metodo");
try {
serverSocket.close();}
catch(IOException e) {
Log.e(TAG, "HiloServidor.cancelarConexion():
Error al cerrar el socket",e);}
}// cancelarConexión()
```

3.1.2 Conexión como Cliente

Para iniciar una conexión como cliente con otro dispositivo, es necesario obtener primero el objeto **BluetoothDevice** que representa al dispositivo remoto, y luego, a partir de él, realizar la solicitud de conexión (*Bluetooth Conectivity*). Los pasos

básicos para implementar el dispositivo Bluetooth como cliente son:

- Usar el **BluetoothDevice** para conseguir un **BluetoothSocket**, y posteriormente llamar a **createRfcommSocketServiceRecord(UUID)** para crear el canal **RFCOMM** de la comunicación. El código **UUID** pasado por este método, debe ser el mismo código **UUID** pasado por el dispositivo servidor cuando el **BluetoothServerSocket** es abierto (*Utilizar el mismo **UUID** es una forma de codificar la cadena **UUID** en la aplicación, que luego será referenciada desde el código del servidor y el cliente)*.
- Iniciar la conexión, llamando el método **connect()**. Una vez que se llama a este método, el sistema realiza una búsqueda **SDP** en el dispositivo remoto con el fin de emparejar el código **UUID**. Si la búsqueda es satisfactoria y el dispositivo remoto acepta la conexión, se establece el canal**RFCOMM**, y el método **connect()** retorna.

Los métodos, y parte del código que implementa este hilo son:

Método HiloCliente(): Con este método se consigue el **BluetoothSocket** que conecta al **BluetoothDevice** de interés haciendo un llamado a **createRfcommSocketToServiceRecord(UUID)**.

```
public HiloCliente(BluetoothDevice dispositivo){
debug("HiloCliente.new()", "Iniciando metodo");
BluetoothSocket tmpSocket = null;
this.dispositivo = dispositivo;
// Obtenemos un socket para el dispositivo con el
que se quiere conectar
try {
 if(x == 0){
       tmpSocket = dispositivo.
          createInsecureRfcommSocketToServi-
ceRecord(UUID.fromString(UUID_INSEGURO));
}
else{
       tmpSocket = dispositivo.
          createRfcommSocketToServiecRecord
(UUID_SEGURO);}
  }// try
catch(IOException e) {
Log.e(TAG, "HiloCliente.HiloCliente(): Error al
abrir el socket", e); }

socket = tmpSocket; }// HiloCliente()
```

Método run(): Es el método encargado de solicitar la conexión al servidor, y de cancelar el proceso de descubrimiento de dispositivos en caso de que este operando, ya que podría causar la pérdida de la conexión.

```
public void run(){
debug("HiloCliente.run()", "Iniciando metodo");
setName("HiloCliente");
if(bAdapter.isDiscovering())
    bAdapter.cancelDiscovery();
try {
socket.connect();
setEstado(ESTADO_REALIZANDO_CONEXION);
x=0;}
catch(IOException e) {
Log.e(TAG, "HiloCliente.run(): " +
"socket.connect(): Error realizando la conexion",e);
      try {
           socket.close();}
      catch(IOException inner) {
            Log.e(TAG, "HiloCliente.run(): " +
           "Error cerrando el socket", inner);}
       conexionFallo();
       x=1;
       return;
   }// catch
// Reiniciamos el hilo cliente, ya que no lo
necesitaremos mas
synchronized(BluetoothService.this){
 hiloCliente = null;}
// Realizamos la conexion
 realizarConexion(socket, dispositivo);}
```

Método cancelarConexion(): este método cancela la conexión que este en progreso, cierra el socket y libera recursos.

```
public void cancelarConexion(){
debug("cancelarConexion()", "Iniciando metodo");
try {
  socket.close();}
catch(IOException e) {
   Log.e(TAG, "HiloCliente.cancelarConexion():
Error al cerrar el socket", e); }}
```

Cada uno de estos comportamientos (*servidor o cliente*) del dispositivo Bluetooth, debe implementarse en otro hilo de ejecución diferente al hilo principal de la aplicación; es decir, se debe crear una clase pública que maneje la conexión Bluetooth de forma paralela a la ejecución de la aplicación, y comunicar los datos mediante un manejador o handler a través de mensajes entre ambos hilos.

3.1.3 Aplicación Bluetooth Cliente-Servidor

La Fig. 2 muestra el funcionamiento de la aplicación desarrollada con una comunicación Bluetooth. Suponga que se dispone de dos dispositivos móviles, el **dispositivo A** que actuará como cliente, y el **dispositivo B** que actuará como servidor. Cuando el **dispositivo A** envía una petición de conexión al **dispositivo B**, este abre un socket de comunicación llamado **BluetoothSocket**, mientras que el **dispositivo B**, que actúa como servidor, abre un socket **BluetoothServerSocket**. Cuando el **dispositivo B** acepta la solicitud de conexión realizada por el **dispositivo A**, este automáticamente abre un **BluetoothSocket** y pasa del hilo servidor al hilo de conexión. Similarmente, el **dispositivo A** que no abre ningún otro socket, sino que continúa con el que había abierto inicialmente,pasa del hilo cliente al hilo de conexión. Una vez que la conexión entre ambos dispositivos se ha establecido, el hilo de conexión es el encargado de administrar el intercambio de datos entre los dos dispositivos y de comunicarlos a la actividad principal a través de un handler. Cuando la comunicación es finalizada, este hilo es el encargado de cerrar los sockets de ambos dispositivos, para iniciar nuevamente el proceso de atender y/o solicitar conexiones.

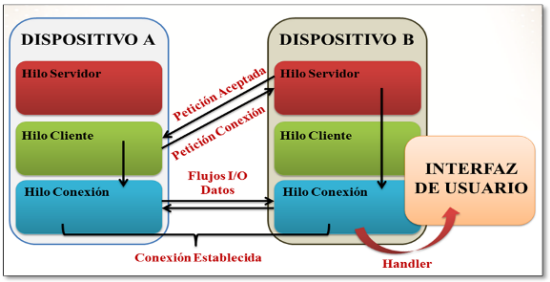

Fig. 2. Solicitud, aceptación y establecimiento de la comunicación Bluetooth entre dispositivos.

3.2 Base de Datos SQLite

Android provee soporte para el manejo de base de datos **SQLite**. Cualquier base de datos que se cree en **SQLite**, puede ser accesada mediante su nombre desde cualquier clase creada dentro de la aplicación, pero no fuera de ella. Para crear una base de datos de este tipo, se crea una subclase de **SQLiteOpenHelper** que inicializa la base de datos con el nombre de la tabla, y la versión de la misma. Cuando la aplicación es cargada por primera vez, el método **onCreate()** es invocado para crear la tabla de la base de datos, y el método **onUpgrade()** es invocado solamente, cuando la versión de la base de datos es incrementada por el desarrollador (*Dinh, 2014*).

La clase pública **DataBaseManager,** es la clase que se encarga de crear la tabla donde se almacenarán los datos. Para efectos de esta aplicación, se crea la tabla **"recorrido"** que se compone de los campos **id, hora, latitud, longitud y fecha**. El campo **id**, es un entero que se incrementa automáticamente, e identifica las filas dentro de la tabla; el resto de los campos de la tabla son los que se manipulan en la aplicación para insertar, modificar y borrar los datos.

3.3 Lista de Datos

Las listas son una colección organizada de datos, donde cada elemento tiene un identificador que permite accesar a un elemento específico. El identificador de la lista inicia en cero y se incrementa automáticamente conforme se van insertando datos. Para crear la vista en forma de lista en Android, es necesario considerar los siguientes elementos:

- **ListView:** Es la clase que representa la vista vertical de datos en Android. Este tipo de vista se encarga de mostrar los datos al usuario, y se compone de un ScrollView que permite realizar un desplazamiento vertical a lo largo de la pantalla, cuando los datos contenidos en la lista exceden el tamaño de la vista de la pantalla del dispositivo móvil.
- **ArrayList<*clase contenedora de datos*>:** Es una implementación de una lista pasada por un arreglo que contiene la información de una "colección específica de datos". Esta *colección especifícada de datos*, es la información que más adelante se incorpora en la lista, y pueden ser del mismo tipo de dato (*long, float, string, etc*) o diferente. Para administrar estos datos, se crea una "clase contenedora de datos" a la cual se le extrae o inserta la información mediante el **ArrayList**.
- **ArrayAdapter<clase contenedora de datos>:**Es una clase que comunica a un **ListView** los datos necesarios para crear las filas de la lista, además de proveer los **Views** para cada elemento de la lista. Los adaptadores se representan por la clase **Adapter**, y varian dependiendo de la naturaleza de los datos. Android ofrece **Layouts** para aplicaciones de listas básicas de uno, dos o hasta tres elementos simples. Para realizar una vista de una lista personalizada, los **Layouts** que ofrece Android no siempre se adaptan a la situación en cuestión, y se hace necesario diseñar un **Layout** propio.

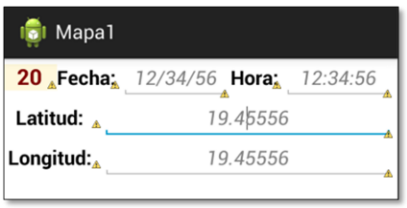

Fig. 3. Layout personalizado de la vista por fila de la lista.

3.4 Mapa de Datos

Para trabajar con mapas en Android, es necesario utiliar el **API** de **Google Maps**. El **API** de Mapas requiere de un **API Key** para poder utilizar los mapas de google. Este **API Key** se genera en la página *www.console.developers.google.com* con la huella digital **SHA1** del proyecto, el nombre del paquete de la aplicación, y la habilitación del **API** de **Google Maps Android API V2**. Este **Key** se referencia en el Android Manifest de la aplicación:

```
<meta-data
android:name="com.google.android.gms.version"
android:value="@integer/google_play_services_version" />
<meta-data
android:name="com.google.android.maps.v2.API_KEY"
android:value="AIzaSyB2qO4SuRUj9IfHkMsWQlZwVj7dZvejysM" />
```

Para insertar un mapa en la aplicación móvil, se coloca un Frame Layout en el fragment de interés, para alojar el mapa (Fig. 4).

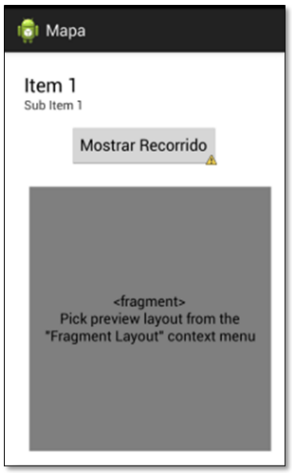

Fig. 4. Fragment en el Layout de la aplicación.

Para usar el **API** de Mapas en la aplicación móvil, es necesario declarar los siguientes permisos en el android Manifest:

```
<uses-permission
android:name="android.permission.INTERNET" />
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
```

4. RESULTADOS

La aplicación móvil implementa una de las nuevas interfaces gráficas diseñadas para el sistema operativo de Android: el Navegation Drawer. Esta interfaz gráfica, separa mediante vistas cada uno de los diferentes bloques de la aplicación, empleando una sola Activity mediante un menú desplegable horizontalmente.

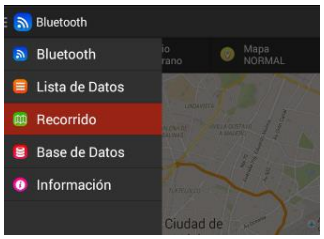

Fig. 5. Menú desplegable de la aplicación.

La vista "Bluetooth" (Fig. 6), permite manejar todas las opciones disponibles por el **API** de Bluetooth de Android con respecto al adaptador local del dispositivo móvil. Esto es, que permite activar y/o desactivar el módulo Bluetooth, iniciar la búsqueda de nuevos dispositivos Bluetooth, mostrar los dispositivos emparejados, establecer y/o cancelar conexiones entre dispositivos Bluetooth con conexiones seguras (con dispositivos móviles Android) o inseguras (con otros módulos Bluetooth), enviar datos en formato de texto a otros dispositivos móviles con Android, solicitar, recibir y almacenar datos de posicionamiento geográfico y salir de la aplicación.

En el fragment que se muestran las *n* posiciones geográficas recibidas del sistema electrónico (Fig. 7), solicita por cada posición los datos de fecha, latitud, longitud y hora almacenados en la base de datos **SQLite** del dispositivo móvil, para darle el formato de **día/mes/año** a la fecha, **hh:mm:ss** al tiempo, y convertir las coordenadas de latitud y longitud de grados, minutos y fracciones por minuto a grados decimales, de acuerdo a como los

interpreta el **API** de Mapas de Android, con el propósito de dar mayor claridad sobre los datos registrados al usuario.

El fragment del "Recorrido" (Fig. 8), permite dibujar en un mapa de google todas las posiciones geográficas almacenadas en la base de datos del dispositivo móvil, identificando el inicio del recorrido con una bandera de color azul, el fin del recorrido con una bandera roja, y las demás posiciones con un ícono en forma de camión de color azul claro, sobre los cuales, al pulsar sobre la pantalla cada uno de ellos, se obtendrá la información de la hora y el número de la posición en la que se encontraba el camión en ese momento con respecto al recorrido realizado.

El *fragment* de "Base de Datos" (Fig. 9), es la vista que informa al usuario las características principales de los datos almacenados, indicando la cantidad de posiciones geográficas almacenadas, la fecha del recorrido realizado, la hora de inicio y la hora de fin del recorrido, borrar y consultar los datos, además de cargar a la aplicación una base de datos experimental, para que el usuario pueda apreciar las funcionalidades de la lista de datos y mapa del recorrido, antes de iniciar con la solicitud y recepción de datos al sistema electrónico.

Para efectos de comprobar la aplicación desarrollada, y la comunicación del dispositivo móvil con el sistema electrónico, se realizó un recorrido en automóvil a través de la ciudad de México, y posteriormente, a través de una comunicación Bluetooth entre el sistema electrónico y el dispositivo móvil, se hizo la solicitud, y recepción de los datos (Fig. 10 y 11).

Fig. 6. Fragment del módulo Bluetooth.

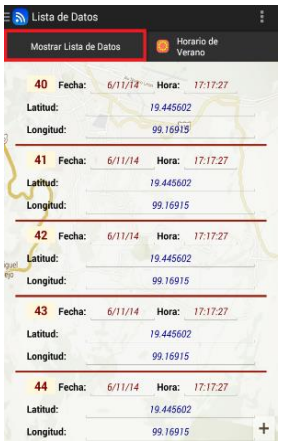

Fig. 7. Fragment de la lista de datos.

Fig. 8. Fragment del mapa de datos

Fig. 9. Fragment de la administración de la base de datos

Fig. 10. Recepción de datos del sistema electrónico en el Sony Xperia T2 Ultra.

Fig. 11. Recorrido realizado en automóvil a través de la ciudad de México.

5. CONCLUSIONES

La aplicación móvil en Android requiere como mínimo la versión 4.0 de Android, 1.2 GHz de velocidad en el procesador, 1 GB de memoria **RAM** y un dispositivo móvil con una resolución de pantalla de 720 x 1280 píxeles (*debido a que la interfaz gráfica está diseñada especialmente para esta resolución de pantalla*) para su ejecución, debido a que la apertura de un socket de comunicación Bluetooth para la transferencia y recepción de datos, consume muchos recursos del dispositivo móvil, haciendo que la ejecución de la aplicación sea lenta.

Se optimizó el desempeño de la aplicación móvil en Android al hacer uso del *Navegation Drawer*, que es un tipo de interfaz gráfica que sólo maneja una actividad, y lanza diferentes vistas para cada función de la aplicación a través de fragmentos.

Al realizar una conexión *Bluetooth* del dispositivo móvil con otro dispositivo *Bluetooth*, la conexión se mantendrá abierta y la transmisión y recepción de datos será posible, siempre y cuando la vista que administra la comunicación, no sea cambiada por otra. Si la vista se minimiza, la comunicación Bluetooth sigue estando activa, pero si se cambia, la conexión se cancela, se interrumpe la comunicación actual, y se tiene que volver a establecer otra conexión Bluetooth para reanudar el intercambio de datos.

Si la recepción de datos en la aplicación móvil se interrumpe de forma inesperada, el almacenamiento de los datos en la base de datos SQLite puede ser erróneo, y las acciones de lanzar las vistas "Lista de Datos" y "Recorrido", generará un error que hará que se cierre la aplicación, siendo necesario, borrar todos los datos almacenados en la base datos, antes de volver a realizar la solicitud y recepción de datos.

RECONOCIMIENTOS

Al centro de Investigación en Ciencia Aplicada y Tecnología Avanzada – **CICATA** unidad Legaria, a la Escuela Superior de Computo – **ESCOM** del Instituto Politécnico Nacional – **IPN** y al Conacyt por el apoyo, asesoría y acompañamiento brindado al desarrollo de este trabajo.

ANEXOS

La información de posicionamiento **GPS** del camión en movimiento, se almacena cada 30 segundos en una memoria **SD CARD** que se encuentra dentro del sistema electrónico. La adquisición de datos **GPS**, se activa y desactiva, de acuerdo al horario de actividad de operaciones del camión de la empresa, con el propósito de ahorrar el consumo de energía proveniente de la batería del camión. El hardware se encuentra compuesto por dos microcontroladores de la empresa Texas Instruments: el **TM4C123GH6PM** como microcontrolador principal, encargado de operar el módulo **Bluetooth**, **GPS** y **OpenLog Data Loger,** y el **MSP430G2553** como microcontrolador secundario, encargado de operar módulo de **Reloj de Tiempo Real**, para activar y desactivar el módulo **GPS**. Las sentencias que se almacenan en la memoria **SD CARD** corresponden a cada posición del recorrido realizado por el camión (Ecuación 1), y tienen un tamaño de 39 bytes (Ecuación 2).

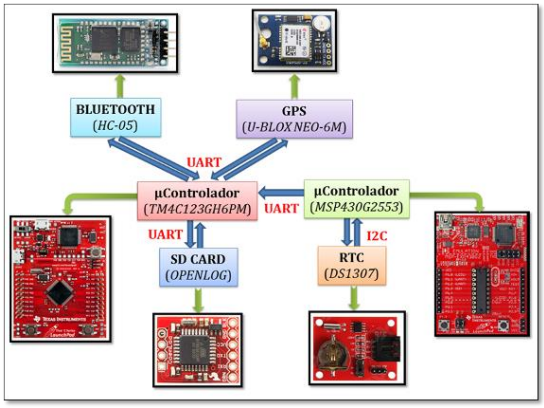

Fig. 12. Diagrama de bloques del Sistema Electrónico.

Hora, Latitud, Longitud, Fecha
021137.00**,** 1926.12622**,** 09912.37385**,** 061114 (1)

9 bytes + 1 byte + 10 bytes + 1 byte + 11 bytes + 1 byte + 6 bytes = 39 bytes (2)

REFERENCIAS

Blanco B. & Romero J. (2010), *Geolocalización, Monitoreo y Rastreo de vehículos y usuarios móviles*, Instituto Politécnico Nacional (IPN), Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas (UPIITA), Ciudad de México, México.

Gonzáles R., Benítez E., Blanco N. & Villapol M. (2012), *Uso de teléfonos inteligentes como una solución de comunicación en aplicaciones de seguimiento de rutas en flotas de vehículos*, Universidad Simón Bolívar, Departamento de Computación y Tecnología de la Información, 5To Congreso iberoamericano de estudiantes de ingeniería eléctrica (V CIBELEC) Caracas, Venezuela.

Cedeño J. (2013). *Desarrollo de aplicación para presentar reportes gráficos (rutas vehiculares) que se visualicen en google maps*, Universidad de Guayaquil, Facultad de Ciencias Matemáticas y Físicas, Ingeniería en Sistemas Computacionales, Guayaquil, Ecuador.

Pacheco L. (2014), *Aplicación GIS para un sistema de control geo -referenciado de pedidos y*

entrega de muebles, Universidad San Francisco de Quito, Colegio de Postgrados, Quito, Ecuador.

Dinh T. (2014), *StudyHive Android Application*, Faculty of the Deparment of Computer Engineering, California State University, Sacramento, United Stated.

Yan M. & Shi H. (2013), *Smarth Living using Bluetooth Base Android Smartphone*, International Journal of Wireless & Mobile Networks (IJWMN), Melbourne, Australia. Vol. 5, No. 1, pp. 65-72.

Pan W., Lou f. & Xu L. (2012), *Research and desing of chatting room system based on Android Bluetooth*, Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on, Yichang, China. pp. 3390-3393.

Chung C., Wang S., Huang C. & Lin C. (2011), *Bluetooth based Android Interactive Applications for Smart Living*, Innovations in Bio-inspired Computing and Applications (IBICA), 2011 Second International Conference on IEEE, Shenzhen, China. pp. 309-312.

Potts J. & Sukittanon S. (2012), Exploting Bluetooth on Android Mobile Devices for Home Security Application, Southeastcon, 2012 Proceedings of IEEE, Orlando Fl, United States. pp. 1-4.

Jeong J. Lim J., Hyun W. & Lee W. (2014), *A Remote Lock System Using Bluetooth Communication*, 2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing IEEE, Seoul, South Korea. pp. 441-446.

Android Developer, http://developer.android.com/index.html. (Consultado: 18 de mayo de 2015).

Bluetooth Connectivity, http://developer.android.com/guide/topics/connectivity/bluetooth.html. (Consultado: 24 de junio de 2015).

IEEE Standars (2005). *Part 15.1: Wireless medium Access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs)*, Revision of IEEE Std: 802.15.1-2002, pp. 19-24, 437-453.

Téllez E., Moreno G. y Antonio L. (2008-2009). *Bluetooth: Aplicabilidad del Estándar IEEE 802.15.1*, Instituto Tecnológico de Zacatepec, Zacatepec, Morelos, México.