libLocation: acceso a dispositivos de localización para gvSIG desktop y mobile

Juan G. Jordán Aldasoro[(1)], *Manuel Planells Jiménez*[(2)]

[(1)] Instituto de Robótica, Universitat de València, jjordan@robotica.uv.es.
[(2)] Instituto de Robótica, Universitat de València, mplanells@robotica.uv.es.

RESUMEN

Inicialmente integrada en el piloto de gvSIG Mobile, la librería libLocation tiene como objetivo dotar a los proyectos gvSIG Desktop y gvSIG Mobile un acceso transparente a fuentes de localización. La librería se fundamenta en las especificaciones JSR-179 -API de localización para J2ME- y JSR-293 -API de localización para J2ME v2.0-, proporcionando una interfaz uniforme a diferentes fuentes de localización, mediante funciones de alto nivel. Asimismo, se extiende la funcionalidad de estas APIs para permitir la gestión de datos específicos del tipo de fuente de localización y el ajuste de parámetros de bajo nivel, además de incorporar métodos de localización adicionales, como la aplicación de correcciones vía protocolo NTRIP. La librería libLocation está actualmente en proceso de desarrollo y será publicada y liberada junto con la versión definitiva de gvSIG Mobile. Junto con libLocation se están desarrollando extensiones que permiten el acceso a esta librería desde gvSIG Desktop y gvSIG Mobile.

Palabras clave: *gvSIG, librería, localización, software libre.*

ABSTRACT

Formerly integrated in the gvSIG Mobile pilot, libLocation library has the goal of providing a transparent access to location sources to the gvSIG Desktop and gvSIG Mobile projects. It is based on the specifications JSR-179 Location API and JSR-293 Location API v2.0, for J2ME, providing a uniform interface to different location sources, through high level methods. This functionality is extended to enable the management of data specific for the type of source and the adjustment of low level parameters, as well as adding additional location methods and providers, like corrections via NTRIP protocol. LibLocation is currently under development and will be published with the definitive version of gvSIG Mobile. Together with the library, extensions to access the library from gvSIG Desktop and gvSIG Mobile are being developed.

Key words: *gvSIG, library, location, open source.*

INTRODUCCIÓN

LibLocation fue ideada con la idea de cumplir los siguientes objetivos, dentro del proyecto gvSIG Mobile [1]:

- Soporte a las plataformas J2SE y J2ME. Dado que gvSIG Mobile funciona sobre plataforma J2ME y perfil CDC (Connected Device Configuration) [2].
- Proporcionar funcionalidad tanto en en el ámbito de la navegación como de la comunidad GIS.
- Soporte a diferentes protocolos y sistemas de localización.
- Interfaz de programación sencilla y de alto nivel.

FUNCIONALIDAD ACTUAL

La versión actual de libLocation proporciona 3 bloques de funcionalidad a gvSIG Mobile.

Ajuste de parámetros de conexión

Actualmente libLocation permite la conexión únicamente con dispositivos GPS a través del protocolo NMEA. Es posible escoger el puerto en el que se conecta el dispositivo GPS, así como el rango binario de esta conexión. La conexión con dispositivos bluetooth es posible a través del puerto serie bluetooth emulado que proporcione el sistema operativo (perfil RFCOMM [3]).

Asimismo es posible conectarse con un dispositivo GPS simulado. Para ello debe disponerse de un fichero log que contenga un volcado de sentencias en el protocolo NMEA recibidas por el puerto serie (o generadas mediante otros programas). A efectos prácticos es como conectarse a un GPS a través del puerto serie, pero en modo offline. Esta funcionalidad es útil para hacer pruebas cuando no hay visibilidad de satélites (en el interior de edificios, en presentaciones y tutoriales, durante la fase de desarrollo de software, etc.).

Figura 1: *Diálogo de ajustes de conexión de gvSIG Mobile*

Visualización de la actividad GPS

El procesado de los diferentes mensajes NMEA del GPS permite obtener varios datos sobre la calidad de la señal:

- **Constelación de satélites:** elevación, azimut y nivel de señal de cada satélite visible. Número de satélites usados en la solución de posicionamiento.
- **Datos de la posición actual:** latitud, longitud, altura, velocidad, orientación, tiempo GPS, parámetros de error (dilución de la precisión), etc.
- **Consola del puerto serie:** volcado de los datos recibidos por el puerto serie, útil para comprobar si existen problemas de conexión.

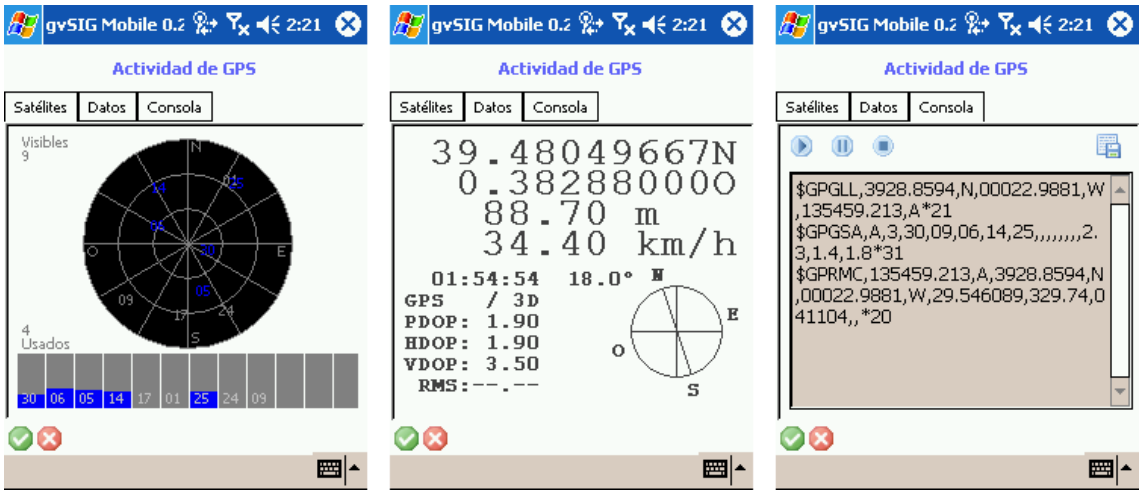

Figura 2: *Diálogos de actividad del GPS de gvSIG Mobile*

Registro de waypoints y tracks

Cuando la aplicación está conectada a un GPS, es posible iniciar, pausar y detener el registro de las posiciones recibidas en un track. Un track tiene la función de registrar todas las posiciones capturadas durante una cierta actividad, por ejemplo, un recorrido en coche. Asimismo se pueden almacenar waypoints a demanda del usuario, cuya función es señalar un punto de interés o un paso de una ruta.

Estos tracks y waypoints se pueden registrar en el formato estándar GPX, y también en un formato sencillo como CSV que puede ser importado fácilmente en otros programas o en hojas de cálculo.

El diálogo de configuración de logs permite ajustar los nombres de archivo de estos ficheros, así como el formato a utilizar. También es posible registrar los datos crudo proveniente del puerto serie, permitiendo genenar logs del protocolo NMEA que posteriormente pueden ser usados para simular un GPS cuando no hay visibilidad de satélites.

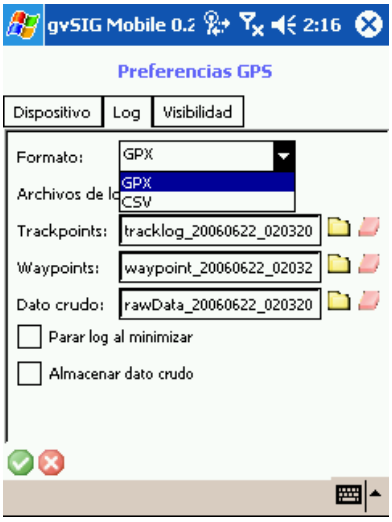

Figura 3: *Diálogo de ajustes de registro de tracks y waypoints*

DETALLES DE DISEÑO

La primera versión de libLocation, que actualmente acompaña al piloto de gvSIG Mobile, se apoya en el paquete org.dinopolis.gpstool, distribuido bajo licencia LGPL junto con la aplicación GPSylon [4]. La última distribución de GPSylon data de octubre de 2006.

La librería libLocation se organiza en dos paquetes principales:

- **org.dinopolis.gpstool.gpsinput:** incluye las clases e interfaces generales de acceso al GPS, proveyendo comunicación con dispositivos serie y dispositivos simulados basados en archivos, y proporcionando un marco para el desarrollo de clases que provean acceso al GPS, incluyendo la implementación del protocolo de comunicación NMEA.
- **org.gvsig.mobile.location:** eleva el nivel de abstracción del acceso a dispositivos GPS, proporcionando una interfaz más sencilla, con respecto a la utilizada en gpsinput. Modifica parte de la funcionalidad ofrecida por gpsinput y añade funcionalidad específica para gvSIG Mobile.

Descripción del paquete org.dinopolis.gpstool.gpsinput

El módulo de acceso a dispositivos GPS fue diseñado para ser independiente del formato de los datos y de la fuente de datos. Los elementos principales de este paquete con respecto a su utilización son los interfaces GPSDevice y GPSDataProcessor. GPSDevice modela fuentes de datos GPS, y proporciona los datos de forma uniforme, independientemente de la fuente. GPSDataProcessor interpreta los datos de una fuente de datos GPS y proporciona la información procesada de forma uniforme, independientemente del protocolo.

Actualmente existen dos realizaciones del interfaz GPSDevice, la clase GPSSerialDevice que proporciona comunicación con receptores GPS a través del puerto serie y la clase GPSFileDevice, que proporciona acceso a archivos de registro de datos GPS, lo que también podemos entender como “GPS simulado”.

En cuanto a la interfaz GPSDataProcessor, la implementación por defecto la proporciona GPSGeneralDataProcessor, una clase abstracta que implementa la

mayoría de los métodos de GPSDataProcessor excepto los más específicos de apertura y cierre del dispositivo, es decir, la funcionalidad básica que la mayoría de clases que implementen GPSDataProcessor deberían utilizar. La clase GPSNmeaDataProcessor, incluida en el paquete org.dinopolis.gpstool.gpsinput.nmea, extiende a GPSGeneralDataProcessor, proporcionando una realización concreta de GPSDataProcessor que interpreta el protocolo NMEA0183.

Las clases que implementan GPSDataProcessor almacenan los últimos datos procesados mediante instancias de las clases GPSPosition y SatelliteInfo, que contienen, respectivamente, información sobre posición (longitud, latitud, altitud) e información sobre la situación de los satélites (código PRN, elevación, azimuth y relación señal a ruido SNR).

Finalmente existen dos interfaces que pueden implementar cualquier clase que quiera recibir información procesada por un GPSDataProcessor: el interfaz GPSRawDataListener, que permite recibir los datos tal cual los envía el dispositivo GPS, y el interfaz GPSDataChangeListener, que debería permitir recibir eventos GPS cuando cambia algún dato procesado del GPS (posición, altitud, velocidad...). Este último interfaz, sin embargo, no está implementado y no se utiliza. En su lugar, las clases que quieran recibir eventos GPS se registran como PropertyChangeListener.

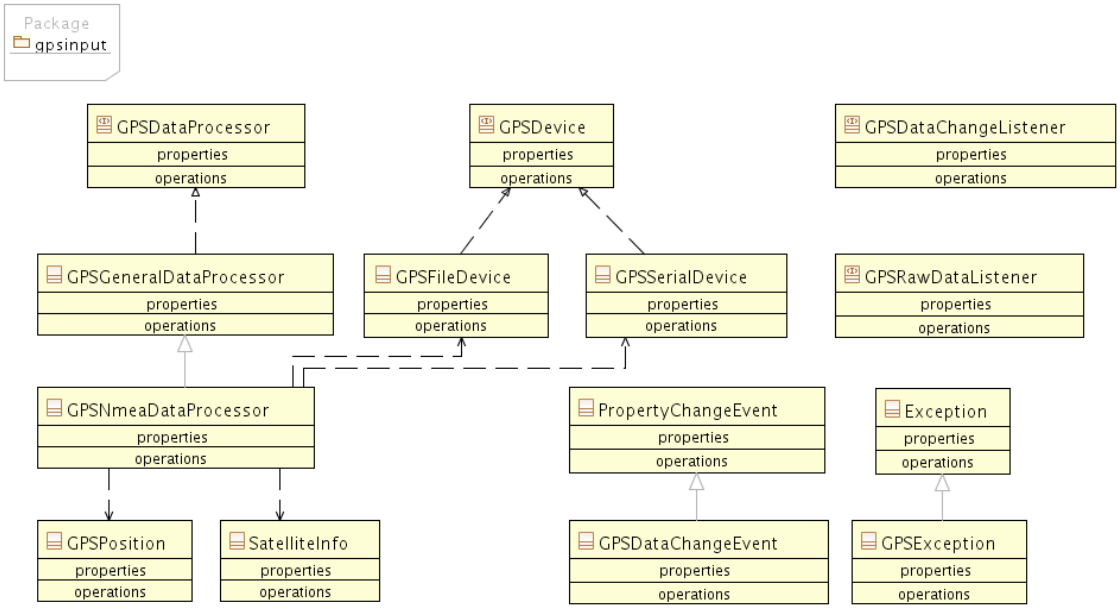

Figura 4: *Diagrama de clase del paquete gpsinput*

Es posible conectar con un GPS directamente mediante el uso de este paquete, aunque no es el objetivo de este artículo realizar un tutorial sobre gpsinput. El siguiente paquete es el que utiliza directamente gvSIG Mobile y que acompañaremos de un ejemplo de uso.

Descripción del paquete org.gvsig.mobile.location

Este paquete se subdivide en varios paquetes:

- **location:** incluye la clase GPSManager, que es la clase desde la que se accede en gvSIG Mobile a las funciones GPS, y la clase GPSFix, que incluye todos los datos calculados por el GPS en un determinado instante.
- **location.listener:** contiene el interfaz que permite a otras clases recibir eventos GPS.

- **location.tracklog:** incluye las clases relacionadas con el registro de waypoints y tracks.

La siguiente figura muestra el diagrama de clase del paquete location, incluyendo algunas clases de otros paquetes.

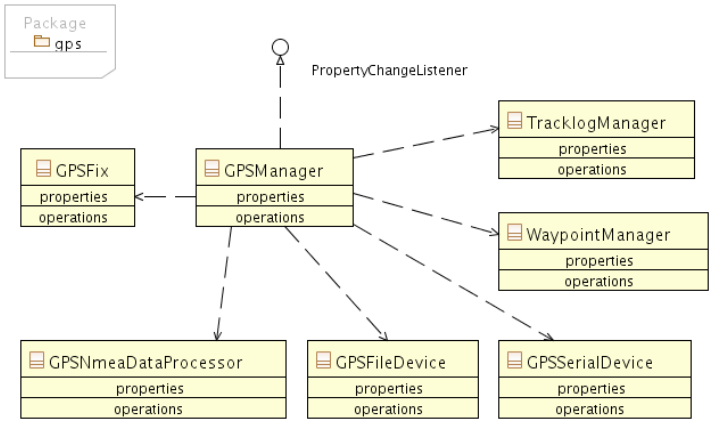

Figura 5: *Diagrama de clase del paquete location*

La clase GPSManager gestiona la creación y configuración de instancias GPSDataProcessor y GPSDevice para el acceso a dispositivos GPS. Asimismo controla el registro de waypoints y tracks en ficheros a través de WaypointManager y TracklogManager.

La clase GPSFix contiene todos los datos pertenecientes a la misma muestra GPS, esto es, calculados en el mismo instante. Estos datos incluyen posición, velocidad, rumbo, posición de los satélites, etc. La utilidad de esta clase actualmente es permitir diferenciar qué datos son recientes y cuáles son más antiguos.

En cuanto al paquete tracklog, incluye las clases que permiten el registro de waypoints y tracks en archivos, permitiendo utilizar el formato GPX o CSV.

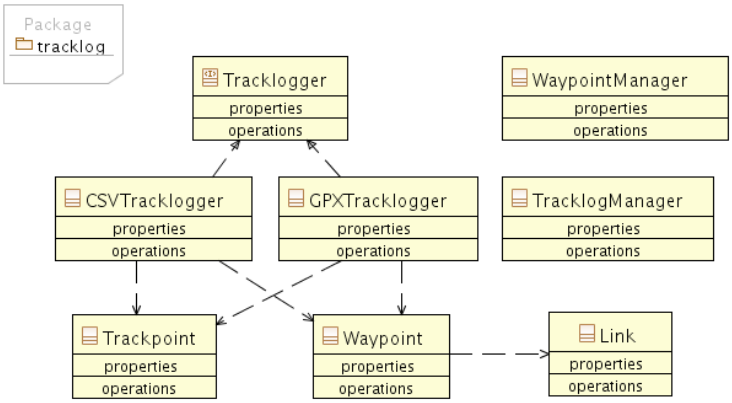

Figura 6: *Diagrama de clase del paquete tracklog*

El interfaz Tracklogger define las operaciones a implementar por las clases que registran waypoints y tracks en archivos, como abrir y cerrar el registro, comenzar a almacenar puntos o detenerse, y almacenar un punto. Las clases GPXTracklogger y CSVTracklogger son implementaciones del interfaz Tracklogger, para el formato GPX y CSV, respectivamente. GPX es un formato de almacenamiento de tracks, rutas y waypoints en XML, muy extendido en el ámbito GNSS.

Las clases Trackpoint y Waypoint definen los datos que puede tener un trackpoint y un waypoint, respectivamente. Cuando el GPSManager recibe un evento de nueva posición, si el TracklogManager está configurado para almacenar puntos, crea una instancia de Trackpoint con los datos de posición y lo envía al TracklogManager para que lo registre en el formato adecuado. Del mismo modo, cuando en la aplicación pulsamos el botón de guardar waypoint, el GPSManager obtiene la última posición calculada e instancia un Waypoint rellenando los datos de posición. El resto de datos los rellena el usuario mediante un diálogo y posteriormente el WaypointManager almacena el waypoint en el formato adecuado.

TracklogManager y WaypointManager son clases estáticas que proporcionan acceso al registro de tracks y waypoints de forma independiente al formato usado. Son las clases que utiliza el GPSManager directamente.

EJEMPLOS DE UTILIZACIÓN

Veamos unos ejemplos de uso de libLocation. Supongamos que queremos incorporar la lectura de datos GPS en una aplicación, a través del puerto serie. Nuestra aplicación va a registrar en un logger la última posición recibida (ver log4j [5]). Para estar seguros de que estamos correctamente conectados al GPS, cuando ejecutemos la aplicación en modo debug, vamos a volcar también en el logger los datos crudos que llegan por el puerto serie.

En primer lugar debemos registrar una clase como listener de eventos GPSRelevantEventsListener (para recibir eventos de posición, entre otros) y una clase GPSRawDataListener (para recibir el evento de datos crudos). Para ello, haremos que esta clase implemente los interfaces GPSRelevantEventsListener y GPSRawDataListener, y más tarde se registrará esta clase como listener de estos eventos desde el GPSManager.

```
public class SimpleCommLocationListener
    implements GPSRelevantEventsListener, GPSRawDataListener {

  // Variable miembro para utilizar el Logger de log4j
  private static Logger logger = Logger.getLogger(SimpleCommLocationListener.class);

  public void gpsNewAltitude(double altitude) {  }

  public void gpsNewHeading(double heading) {  }

  public void gpsNewPDOP(double pdop) {  }

  public void gpsNewPosition(double longitude, double latitude) {  }

  public void gpsNewQuality(int quality) {  }

  public void gpsNewRMS(double rms) {  }

  public void gpsNewSatsUsed(int satUsed) {  }

  public void gpsNewStarfire(int starfire) {  }

  public void gpsNewStatus(int status) {  }

  public void gpsRawDataReceived(char[] buffer, int offset, int length) {  }
}
```

De los métodos que es obligado implementar para estos interfaces, rellenaremos gpsNewPosition y gpsRawDataReceived. En el primero le decimos al logger que muestre la latitud y longitud formateadas. En el segundo, sólo si estamos en modo debug, que muestre cada línea NMEA recibida por el puerto serie.

```
public void gpsNewPosition(double longitude, double latitude) {
  String northing, easting;
  if (longitude >= 0)
    easting = "W";
  else
    easting = "E";
  if (latitude >= 0)
    northing = "N";
  else
    northing = "S";
  logger.info("Nueva posición: " + latitude + northing +
              ", " + longitude + easting);
}

public void gpsRawDataReceived(char[] buffer, int offset, int length) {
  logger.debug("Nuevos datos NMEA recibidos: " + new String(
               buffer, offset, length));
}
```

En el método main de la aplicación (o en el método que consideremos oportuno) realizamos la detección de puerto serie, y en este caso escogemos el primer puerto serie detectado. Posteriormente seleccionamos en el gpsManager conexión por puerto serie, especificamos el nombre de puerto y registramos nuestra clase receptora de eventos. Finalmente tratamos de conectar con el GPS y –para este ejemplo– hacemos un bucle infinito para esperar a que se reciban los eventos GPS.

```
public static void main(String[] args) {
  // Configuración del logger
  setup();
  // Detección del primer puerto serie del sistema
  String portName = null;
  Enumeration portList = CommPortIdentifier.getPortIdentifiers();
  while (portList.hasMoreElements()) {
    CommPortIdentifier portId = (CommPortIdentifier) portList.nextElement();
    if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
      portName = portId.getName();
      logger.info("Usando el primer puerto serie detectado: " + portName);
      break;
    }
  }
  if (portName != null) {
    GPSManager gpsManager = GPSManager.getGPSManager();
    // Selección de dispositivo: puerto serie
    gpsManager.setDeviceType(GPSManager.SERIAL_PORT_DEVICE);
    // Selección de nombre de puerto serie
    gpsManager.setPortName(portName);
    // Selección de rango binario
    gpsManager.setBaudRate(9600);
    // Instancia la clase que recibe los eventos
    SimpleSimLocationListener listener = new SimpleSimLocationListener();
    // Registra esta clase como receptora de eventos de localización
    gpsManager.addGPSRelevantEventsListener(listener);
    // Registra esta clase como receptora del evento de datos crudos
    gpsManager.addGPSRawDataListener(listener);
    try {
      // Conectar al GPS
      gpsManager.connectToGPS();
    } catch (GPSException e) {
      System.err.println("ERROR al conectar al GPS: " + e.getMessage());
    }
  }
  // Esperar a eventos del GPS indefinidamente
  while (true) {}
}
```

Sólo faltaría añadir el método que inicializa el logger y selecciona el modo INFO o DEBUG para el logger, en función del tipo de mensajes que queramos registrar mediante el logger.

```
private static void setup() {
  try {
    PatternLayout l = new PatternLayout("%5p [%t] - %m%n");
    FileAppender fa = new FileAppender(l, LOG_PATH, false);
    // Logger.getRootLogger().setLevel(Level.DEBUG); // Para registrar en modo DEBUG
    Logger.getRootLogger().setLevel(Level.INFO); // Para registrar en modo INFO
    Logger.getRootLogger().addAppender(fa);
  } catch (Exception ex) {
    System.err.println("Error al inicializar el logger: " + ex.getMessage());
  }
}
```

DESARROLLO ACTUAL Y EVOLUCIÓN

libLocation está sufriendo una reestructuración completa que le debe permitir superar algunas limitaciones detectadas, así como añadir nuevas funcionalidades.

Algunas de las limitaciones detectadas en la versión actual basada en el paquete org.dinopolis.gpstool.gpsinput son:

- **gestión de la conexión/desconexión de dispositivos:** gpsinput no genera eventos cuando se desconecta un dispositivo o se produce algún error en la entrada por puerto serie.
- **gestión de configuraciones:** la versión actual no almacena la última configuración utilizada. Sería deseable poder almacenar y recuperar configuraciones.
- **tiempo de proceso del protocolo NMEA:** se ha comprobado que en algunos dispositivos móviles, la lectura y procesado de NMEA por el puerto serie puede llegar a sobrecargar el sistema. Se busca la optimización de este proceso.
- **extensibilidad poco flexible:** cuando se desea añadir funcionalidad es necesario recompilar y generar de nuevo el JAR de la librería. Sería deseable poder añadir funcionalidad en forma de plugins externos.

La nueva versión de libLocation se plantea superar estas limitaciones. Se han tomado las siguientes decisiones de diseño:

- **arquitectura de plugins:** libLocation permitirá el registro de nueva funcionalidad en forma de plugins de la propia librería.
- **uso de un API conocida:** se basará en el API de localización JSR179 [6], cuya interfaz es conocida por la comunidad de desarrolladores, y contempla mecanismos de cambios en la disponibilidad del GPS que son deseables para nuestro desarrollo. La API será extendida para dar soporte a la funcionalidad que ésta no proporciona y gvSIG Mobile necesita.
- **nueva funcionalidad:** se añadirán nuevos protocolos de conexión a dispositivos de posicionamiento (TSIP, SIRF, gpsd, RTCM/NTRIP); búsqueda automática de dispositivos y gestión de configuraciones; se creará un almacén de waypoints, tracks y rutas que permitirá importar y exportar estos datos a la aplicación, para utilizarlos en la navegación; filtrado y promediado de posiciones en la captura de waypoints y tracks; generación de alertas de navegación y calidad de la señal de posición.

- **optimización de procesos:** para plataformas más limitadas –PDAs antiguas y teléfono móviles– se realizarán implementaciones más eficientes con respecto al parseado de datos.

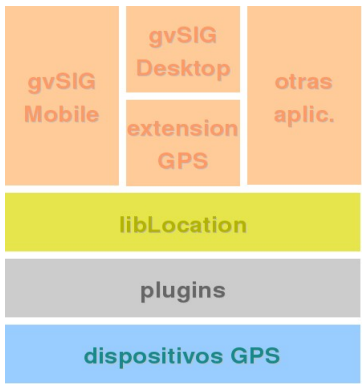

Figura 7: *Arquitectura de plugins para libLocation*

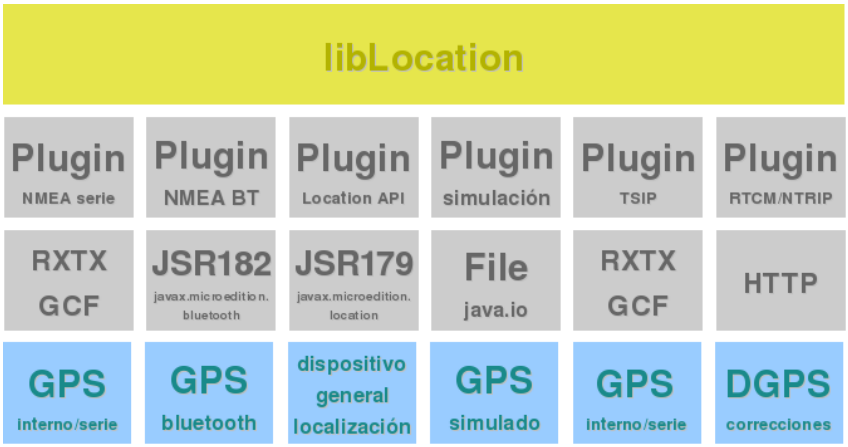

Figura 8: *conexión a diferentes dispositivos a través de plugins*

CONCLUSIONES

libLocation proporciona a gvSIG Mobile funciones de localización a través de una interfaz de alto nivel. Al ser concebida como una librería independiente, es posible utilizar libLocation desde gvSIG Desktop y también desde cualquier aplicación Java.

La librería se encuentra en proceso de desarrollo de una nueva versión que proporcionará nueva funcionalidad y corregirá las limitaciones actuales detectadas, aspirando a convertirse en una librería general de acceso a dispositivos de localización con posibilidad de extensión por medio de plugins, y uso a través de una API conocida como es la JSR179 de Localización.

REFERENCIAS

- Página del proyecto gvSIG Mobile. http://www.gvsig.org/web/projects/gvsig-mobile, http://www.gvsig.gva.es/index.php?id=gvsig-mobile
- Java ME Technology - CDC. http://java.sun.com/javame/technology/cdc/index.jsp
- Perfil Bluetooth RFCOMM (Radio Frequency Communication) http://en.wikipedia.org/wiki/Bluetooth_protocols

- GPSylon. http://www.tegmento.org/gpsylon/
- Apache log4j. http://logging.apache.org/log4j/
- JSR 179: Location API for J2ME. http://jcp.org/en/jsr/detail?id=179