Nueva librería de acceso a dispositivos de localización libLocation

Juan Guillermo Jordán Aldasoro, Instituto de Robótica, Universidad de Valencia, España
Manuel Planells Jiménez, Instituto de Robótica, Universidad de Valencia, España

Resumen

La librería libLocation proporciona a gvSIG Mobile funciones de localización a través de una interfaz de alto nivel. Se fundamenta en las especificaciones JSR 179 –API de localización para J2ME– y JSR 293 –API de localización para J2ME v2.0–, que proporcionan una interfaz uniforme a diferentes fuentes de localización. Al ser concebida como una librería independiente, es posible utilizar libLocation también desde gvSIG Desktop, así como desde cualquier aplicación Java.

La librería está en proceso de desarrollo, siendo actualmente funcional la comunicación con dispositivos GPS a través de protocolos NMEA, SiRF, TSIP y GPSd. Además, la librería soporta funcionalidades de localización de precisión como la conexión a servidor NTRIP para el envío de correcciones RTCM a dispositivos GPS o el registro de observables en ficheros RINEX para la aplicación de correcciones en postproceso.

libLocation también incluye funciones de navegación, como la gestión de waypoints, tracks y rutas, con importación y exportación a formatos GPX y CSV, así como el disparo de alertas de proximidad.

Se ofrecen también diferentes posibilidades para la captura de puntos, como el promediado de muestras y el filtrado de muestras por distancia, velocidad y orientación.

Palabras clave: **localización, GPS, librería, gvSIG**

1 Objetivos

El principal objetivo de nuestro trabajo es el **desarrollo de una librería de localización que de soporte a los proyectos gvSIG Mobile y gvSIG Desktop**.

Otros objetivos que debe cumplir LibLocation incluyen:

- **Soporte a las plataformas J2SE y J2ME CDC.** Dado que gvSIG Mobile funciona sobre plataforma J2ME y perfil CDC (Connected Device Configuration), este debía ser el perfil Java mínimo a soportar. La compatibilidad con J2SE es prácticamente directa, ya que J2SE contiene a J2ME CDC, salvo algunos paquetes.
- **Proporcionar funcionalidad tanto en en el ámbito de la navegación como de la comunidad GIS.** gvSIG Mobile es una aplicación flexible, pensada para diversos perfiles de usuarios, que abarcan desde el aficionado que desea utilizarla para navegación en montaña, hasta el profesional que necesita posicionamiento preciso y soporte a protocolos avanzados.
- **Soporte a diferentes protocolos y sistemas de localización.** La librería debe soportar protocolos de comunicación con dispositivos de localización del ámbito de consumo, como el protocolo NMEA, así como protocolos del ámbito profesional como TSIP. Se aspira a que también se incorporen en un futuro otros sistemas de localización además de GPS de modo que las interfaces deben estar preparadas para ello.

- **Soporte de almacenamiento de datos para waypoints, tracks y rutas.** La librería debe proporcionar funcionalidad de navegación mediante waypoints y rutas, así como registro de tracks. Los datos deben ser exportables e importables de ficheros en formatos de uso común, como GPX y CSV.
- **Interfaz de programación sencilla y de alto nivel.** La librería debe ser extensible por otros desarrolladores, y también debe poder ser usada en otras aplicaciones independientes del proyecto gvSIG. Para ello se debía escoger una API sencilla y que expusiera funciones de alto nivel.
- **Extensibilidad.** La librería debe ser extensible de forma sencilla y flexible, permitiendo diferentes empaquetados con tamaños variables de fichero JAR.

2 Metodología

Nuestra metodología se ha basado basa en los siguientes principios:

- **Uso de JSRs** donde sea posible. Como medida de reutilización de código y muy importante a la hora de aligerar nuestra aplicación en plataformas móviles. Algunas especificaciones utilizadas incluyen la JSR 82 de Bluetooth y la JSR 179 de localización.
- Uso de **java preferiblemente ante soluciones JNI**. En este caso el objetivo es garantizar la portabilidad a todas la plataformas posibles y facilitar el mantenimiento del código y la trazabilidad de errores.
- **Reutilización de código** de otras librerías y proyectos de código abierto, respetando las licencias. Para nuestro desarrollo hemos utilizado código fuente de otros proyectos Java, que generalmente hemos tenido que modificar o integrar en nuestro código, parcialmente o aprovechado sólo algunas ideas. También hemos recurrido a algunos proyectos en lenguaje C.
- **Optimización de recursos.** Las primeras versiones de libLocation sufrían algunos problemas de rendimiento en dispositivos móviles. Se ha hecho hincapié en la optimización de los procesos que se repiten muchas veces por segundo, aunque sean aparentemente sencillos, como el procesado de cadenas y datos binarios en la lectura de protocolos. También se ha tratado de controlar la proliferación de objetos que el recolector debe eliminar –un proceso que no siempre funciona como debiera en dispositivos móviles– y de hilos de ejecución.
- Amplio uso de **tests unitarios**. Importante para controlar la aparición de bugs por pequeños cambios en el código fuente, difíciles de detectar de otra manera. En la medida de lo posible se ha tratado de crear tests unitarios al tiempo que se crea o añade funcionalidad a una clase.
- **Comentar y crear el Javadoc durante el desarrollo** (Comment As You Code). Evita tener que tomar el tiempo de documentar después, ayuda a aclarar ideas antes de escribir el código y reduce los bugs.
- En general, se sigue la **metodología de desarrollo recomendada para gvSIG 2.0**, utilizando el patrón Locator para librerías.

3 La librería

La interfaz se basa en el API de localización JSR 179, cuya interfaz es conocida por la comunidad

de desarrolladores. La API ha sido extendida para dar soporte a la funcionalidad que ésta no proporciona y gvSIG Mobile necesita, que incluye el procesado de datos específicos del tipo de fuente de localización –constelación de satélites y otros datos avanzados que algunos protocolos exponen–, la configuración de parámetros de bajo nivel del dispositivo de localización y otras funcionalidades que veremos en los siguientes apartados.

3.1 Proveedores de localización

La API JSR 179 especifica que el acceso a dispositivos de localización se realizará a través de proveedores de localización (*LocationProvider*). Veamos cuáles son los métodos que expone la JSR 179 en comparación a los que expondrá libLocation.

NOTA: en adelante, todas las interfaces se presentan no tipadas, para simplificar.

API JSR 179	API libLocation
==========	==========
	addAreaListener()
	addLocationListener()
	addMovementListener()
	addPropertyChangeListener()
static addProximityListener()	addProximityListener()
	getCurrentTimestamp()
static getInstance()	---
static getLastKnownLocation()	---
	getLastLocation()
getLocation()	getLocation()
	getLocationProperty()
getState()	getState()
	removeAreaListener()
	removeMovementListener()
	removeLocationListener()
	removePropertyChangeListener()
static removeProximityListener()	removeProximityListener()
reset()	reset()
setLocationListener()	---
	setMovementListener()
	removeMovementListener()
	startAveraging()
	stopAveraging()
	cancelAveraging()

La JSR 179 define un método estático de factoría *getInstance()* que permite obtener un proveedor de entre todos los implementados de acuerdo a ciertos criterios de precisión, coste y consumo eléctrico. Dicho procedimiento resulta limitado para la configuración de dispositivos de localización externos, donde es necesario poder elegir entre los diferentes puertos serie disponibles, velocidades binarias, direcciones Bluetooth de varios dispositivos al alcance, protocolos de comunicación disponibles, etc.

Para hacer posible la configuración de parámetros de bajo nivel de los dispositivos de localización y con objeto de facilitar la extensibilidad de la librería **se ha creado el gestor de proveedores de localización (LocationProviderManager)**, por lo que de momento el método de factoría desaparece –a menos que más tarde se decida implementarlo por compatibilidad.

Se han creado varios listeners nuevos, que veremos en detalle más tarde. El registro de listeners deja de ser estático, ya que deseamos poder utilizar varios proveedores de localización al mismo tiempo, cada uno con sus listeners. El LocationListener deja de ser único, por lo que varias clases se podrán suscribir a eventos de localización. En general, **se han eliminado los métodos estáticos o sustituido por métodos de instancia**, para permitir diferentes implementaciones a partir de una API común, además de permitir el testeo unitario sobre estos métodos.

En general, la mayoría de proveedores de localización van a leer datos de un InputStream. Algunos deben tratar los datos como texto plano –por ejemplo los proveedores que usan el protocolo NMEA y el protocolo GPSd– mientras que otros los tratan como datos binarios –protocolos SiRF y TSIP–. Se ha desarrollado una clase *BufferedStreamReader* que toma el InputStream y alimenta el proveedor mediante líneas de texto o buffers de datos binarios, controlando si el stream se desconecta eventualmente. **Este esquema proporciona una gran flexibilidad, permitiendo que los providers puedan leer los datos de un puerto serie, una conexión Bluetooth, un socket o un fichero, indistintamente**.

Las diferentes formas de conectar a un proveedor de localización implementadas incluyen:

- **Conexión a puertos serie a través de la librería RxTx.** Se trata de una implementación libre de la CommAPI de Sun. Permite acceder a puertos RS232 y USB desde plataformas Windows, Linux y Mac OS. También a dispositivos Bluetooth mediante puerto serie virtual (perfil RFCOMM de la especificación Bluetooth).
- **Conexión a puertos serie a través de la GCF (Generic Connection Framework).** Permite acceder a puertos RS232 desde plataformas Windows Mobile.
- **Conexión a dispositivos Bluetooth a través de la librería Bluecove.** Se trata de una implementación libre de la JSR 82. Permite acceder a dispositivos Bluetooth desde plataformas Windows, Windows Mobile, Linux y Mac OS.
- **Conexión a ficheros.** Permite acceder a ficheros de registro de datos crudos, desde cualquier plataforma. La lectura de fichero permite simular una conexión a una trama de datos registrada previamente, sin necesidad de salir al exterior cada vez que se necesita hacer pruebas.
- **Conexión a demonio GPSd a través de socket.** Permite acceder en tiempo real a los datos crudos de un dispositivo de localización gestionado por GPSd. Este dispositivo puede indistintamente estar conectado a nuestra máquina o a una máquina remota.

En el diseño de los proveedores de localización se ha buscado la flexibilidad y extensibilidad. Se ha explotado el hecho de que la mayoría de protocolos de comunicación con dispositivos de localización funcionan mediante el envío de mensajes, cada uno de los cuales suele tener un identificador asociado.

La estrategia consiste en definir un decodificador general –por ejemplo, el *NmeaSentenceDecoder* para el protocolo NMEA– al que se llama cada vez que se detecta un mensaje. Este decodificador debe ser extendido para implementar todos los mensajes que se desea decodificar. En tiempo de ejecución es necesario registrar los decodificadores de los mensajes que se desea que estén disponibles, y posteriormente se puede activar o desactivar cada tipo de mensaje por separado. **Esto nos permite ignorar los mensajes que no nos interesan, liberando al procesador de una carga innecesaria**. Asimismo **es posible extender la funcionalidad de un determinado protocolo** desarrollando nuevos decodificadores para mensajes que ahora no se procesan.

3.2 Registro de proveedores

Ya se ha mencionado un cambio en la forma de instanciar proveedores de localización respecto a la JSR 179. Vemos en la siguiente figura la API del LocationProviderManager, el gestor de dispositivos de localización, que va a permitir el registro y configuración de proveedores.

```
discoverPlugins()
getDescription()
getLastKnownLocation()
getLocationMethod()
getLocationProvider()
getLocationProviderNames()
isLocationMethodSupported()
registerLocationProvider()
setConfiguration()
unregisterLocationProvider()
unregisterAll()
```

La filosofía de libLocation se basa en el **registro de proveedores de localización.** Esto proporciona una gran flexibilidad ya que, de inicio, la librería no incluye ningún proveedor de localización –permitiendo generar un JAR tan ligero como se desee–. En tiempo de ejecución se indica a la librería qué proveedores de localización estarán disponibles –métodos *registerLocationProvider* y *unregisterLocationProvider*–, **pudiendo incluso incluir proveedores que se encuentren empaquetados en otros ficheros JAR** –método *discoverPlugins*.

En cualquier punto de la aplicación es posible preguntar al gestor de proveedores cuáles proveedores están disponibles –método *getLocationProviderNames*–, detalles de un cierto proveedor –métodos *getDescription*, *getLocationMethod*, e *isLocationMethodSupported*–, obtener una instancia de un proveedor –método *getLocationProvider*– o preguntar cuál ha sido la última localización calculada por cualquiera de los proveedores activos –método *getLastKnownLocation*–. La configuración del proveedor se realizará a través de un método que permita el envío de parámetros al proveedor –método *setConfiguration*.

3.3 Listeners

Se han añadido nuevos listeners a la interfaz de la JSR 179. Algunos responden a eventos simples como la actualización de la posición u otras propiedades, mientras que otros son más complejos, como eventos lanzados cuando el dispositivo se aleja una determinada distancia de la última posición enviada.

En primer lugar, se soportan los listeners ya definidos en la JSR 179. Se trata del *LocationListener* y el *ProximityListener*. El **LocationListener** envía actualizaciones de localización y del estado del proveedor. Las actualizaciones de localización son periódicas mientras que los cambios de estado son puntuales. Se ha modificado su interfaz para permitir el registro de varios listeners a un mismo proveedor, de modo que varios objetos puedan recibir los eventos. En cuanto al **ProximityListener**, permite recibir eventos de proximidad a unas coordenadas definidas.

El **AreaListener** es un nuevo listener introducido en la JSR 293 (versión 2.0 de la API de localización para J2ME) que viene a extender la funcionalidad del ProximityListener, permitiendo en este caso definir áreas para las cuales se desea comprobar la proximidad, la entrada o la salida.

```
areaEvent()
```

```
monitoringInfo()
monitoringStateChanged()
```

El evento más interesante de esta interfaz es areaEvent, que informa de un cruce de la frontera del Área registrada. Una vez se cruza el área es necesario registrar el listener de nuevo, si se quieren recibir nuevos eventos.

El **AveragingListener** permite solicitar el promediado de varias muestras, antes de recibir una actualización de localización.

```
averagingFinished()
averagingProcess()
averagingCancelled()
```

El promediado es asíncrono y se inicia con el método *startAveraging* de *LocationProvider*. Durante el proceso de promediado se envían eventos *averagingProcess*, que permiten conocer la localización promediada parcial, que incluye entre otros valores una estimación de la prercisión obtenida hasta el momento. Esto permite detener el promediado si en algún momento se ha conseguido la precisión deseada, mediante el método *stopAveraging* de *LocationProvider*. El método *cancelAveraging* permite cancelar el promediado en cualquier momento, en cuyo caso el listener recibe un evento *averagingCancelled*. Si no se detiene el promediado, cuando se alcanza el número de muestras promediadas deseado se recibe un evento *averagingFinished*. Una vez termina el promediado es necesario registrar el listener de nuevo, si se quieren recibir nuevos eventos.

El **MovementListener** envía actualizaciones de localización cuando se cumple una combinación de reglas preestablecidas, que incluyen una mínima variación en la distancia, velocidad y orientación. Diseñado para filtrar coordenadas que aportan poca información sobre el movimiento del dispositivo, es adecuado para procesos que necesiten almacenar o procesar sólo la información más relevante para describir el movimiento del dispositivo, por ejemplo, el registro de la traza de un vehículo.

```
movingLocation()
discardedLocation()
```

El listener se registra mediante el método *setMovementListener* de *LocationProvider* –sólo puede haber un *MovementListener* para cada LocationProvider– es permanente hasta que se desregistra, mediante *removeMovementListener*. Cada muestra de localización procesada se envía al listener, bien a través del evento *movingLocation*, si la muestra se considera válida –contiene información de movimiento según los parámetros de configuración– o *discardedLocation*, si se rechaza por redundante.

La figura 1 muestra cómo este listener puede permitir el filtrado de posiciones que aportan poca información, monitorizando los cambios de distancia, velocidad u orientación.

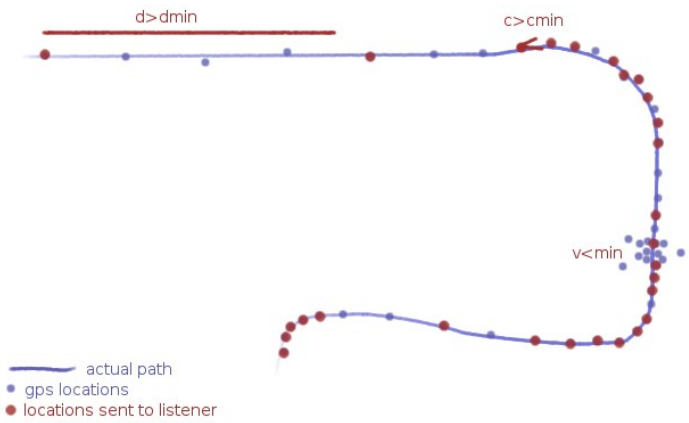

Ilustración 1: Filtrado de posiciones mediante el MovementListener (fuente propia)

Por último, se define el **PropertyChangeListener** para la actualización de cualquier tipo de propiedad que un proveedor de localización sea capaz de procesar. Existen muchas propiedades que cualquier dispositivo de localización envía a través de los diferentes protocolos disponibles –como la constelación de satélites, separación del geoide y el elipsoide, etc.– y para las cuales la JSR 179 no prevé ninguna forma de envío. Esto nos permite la flexibilidad de poder desarrollar más adelante cualquier tipo de proveedor, y ser capaces de comunicar a la aplicación las propiedades procesadas por este.

3.4 Gestión de waypoints, rutas y tracks

La JSR 179 también prevé un almacén de landmarks –también llamados waypoints, y que consisten en un punto con metadatos asociados–. Se trata del **LandmarkStore**, y aporta funciones de búsqueda y filtrado de landmarks. En este caso la API resultaba incompleta para nuestras necesidades, dado que la JSR 179 no prevé el almacenamiento de rutas o tracks. Por ello **se ha extendido su funcionalidad para permitir almacenar rutas** –lista ordenada de waypoints– y **se ha creado un almacén de tracks** –lista ordenada de puntos–, **el TrackStore**.

El desarrollo se ha realizado pensando en que en un futuro pueda haber diferentes implementaciones del LandmarkStore y el TrackStore, por ejemplo, implementaciones basadas en diferentes tipos de base de datos. La implementación actual almacena los datos en una base de datos **HSQLDB**, una implementación ligera de JDBC desarrollada completamente en Java.

La implementación permite la exportación e importación de datos a formatos GPX y CSV, pudiendo ser extendida para dar soporte a otros formatos (como el formato LMX de Nokia).

3.5 Otras funcionalidades

Existen ciertas funcionalidades ya desarrolladas que están en fase de testeo y para las cuales aún no se ha definido una interfaz dentro de libLocation. Estas funcionalidades incluyen la **conexión a servidor NTRIP** y reenvío de correcciones a un dispositivo GPS con capacidad de aplicar correcciones en formato RTCM y el **registro de observables en ficheros RINEX** para la aplicación de correcciones diferenciales en postproceso.

4 Estado de desarrollo actual

A nivel de librería, el estado de desarrollo es del **100% de lo planificado**. En la actualidad son funcionales los siguientes proveedores de localización:

- Proveedor de localización a través de protocolo NMEA.
- Proveedor de localización a través de protocolo SiRF.
- Proveedor de localización a través de protocolo TSIP.
- Proveedor de localización a través de demonio GPSd.

Adicionalmente, la librería soporta la siguiente funcionalidad:

- Conexión a servidor NTRIP y reenvío de correcciones a dispositivo GPS.
- Registro de observables en ficheros RINEX.
- Registro de datos crudos del dispositivo para su posterior uso en simulación.
- Gestión de waypoints, tracks y rutas. Importación y exportación a formatos GPX y CSV.

A nivel de integración con la aplicación mediante la extensión GPS, el estado de desarrollo es del **60% de lo planificado**.

5 Proveedores desarrollados

En esta sección repasamos los proveedores de localización ya desarrollados, su nivel de funcionalidad actual y las posibilidades que nos ofrecen.

5.1 Proveedor de localización a través de protocolo NMEA 0183

Su nombre responde a las siglas de la "National Marine Electronics Association". Se trata del **protocolo más común para la conexión con dispositivos GPS**. Soportado por casi el 100% de los dispositivos del mercado. El protocolo NMEA utiliza una velocidad binaria bastante lenta, generalmente de 4800 o 9600 baudios, por lo cual sólo permite intercambiar información bastante básica, aunque **da soporte a los datos que la mayoría de usuarios no profesionales necesitan**. Estos incluyen la posición, velocidad y tiempo, orientación, constelación de satélites, altitud y separación del geoide, así como algunos datos sobre la precisión de la muestra.

El proveedor NMEA permite escoger los mensajes NMEA que son procesados y los que son ignorados, para mejorar el rendimiento en PDAs. Actualmente se han desarrollado decodificadores para los mensajes NMEA estándar más comunes y algunos mensajes propietarios:

- **GPDBT.** Proporciona datos de la profundidad del receptor bajo el nivel del mar.
- **GPGGA.** Tiempo, latitud, longitud, altitud, separación del geoide, etc.
- **GPGLL.** Tiempo, latitud, longitud.
- **GPGSA.** Satélites activos.
- **GPGST.** Desviación RMS del error de la muestra, en latitud, longitud, altitud.

- **GPGSV.** Satélites a la vista, PRN, elevación, azimuth y C/N0 (relación portadora a ruido).
- **HCHDG.** Mensaje propietario Garmin. Orientación, desviación y variación magnética.
- **GPHDT.** Orientación.
- **GPRMC.** Tiempo, latitud, longitud, velocidad y orientación.
- **PGRME.** Mensaje propietario Garmin. Error estimado horizontal (HPE), vertical (VPE) y esférico equivalente (EPE).
- **GPVTG.** Velocidad y orientación.

Como se puede comprobar con un simple repaso a la descripción de los mensajes anteriores, desafortunadamente no existe ningún mensaje NMEA que incluya los datos mínimos de navegación –latitud, longitud, altitud, velocidad, orientación y tiempo– de manera atómica, por lo que normalmente deberemos suscribirnos a varios mensajes. Por otro lado, la librería realiza estimaciones de la velocidad y la orientación cuando estos datos no son enviados por el dispositivo.

5.2 Proveedor de localización a través de protocolo SiRF

El protocolo SiRF es un protocolo binario desarrollado por la empresa del mismo nombre para la **comunicación con dispositivos GPS que incorporan alguno de sus chipset –SiRF Star II, SiRF Star III...–**. Suele utilizar un rango binario mayor al utilizado en el protocolo NMEA, a partir de 19200 baudios en este caso, que permite el intercambio de una mayor cantidad de datos. La mayoría de dispositivos con tecnología SiRF también incorporan el protocolo NMEA, aunque existen modelos que no lo hacen.

SiRF **proporciona datos mucho más detallados** que NMEA sobre el estado del receptor, incluyendo el estado de la sincronización de cada uno de los canales del receptor, datos crudos recibidos de la constelación de satélites GPS, datos de almanaque y efemérides, etc. Además SiRF incluye en un sólo mensaje todos los parámetros relevantes para la navegación (mensaje SiRF 41).

Otro uso interesante es el modo de librería de navegación. Para entrar en este modo es necesario enviar previamente un mensaje al receptor. En modo librería de navegación, se activan los mensajes 28, 29, 30 y 31 –dependiendo del modelo sólo se activarán algunos– que proporcionan datos avanzados incluyendo **observables como el pseudorrango, fase de la portadora, y datos de la posición exacta de los satélites**. Estos datos pueden ser almacenados en ficheros **RINEX** para la corrección diferencial en postproceso, y potencialmente podrían calcularse correcciones diferenciales en tiempo real.

Actualmente se han desarrollado decodificadores para los siguientes mensajes SiRF:

- **Mensaje 2. Measure Navigation Data Out.** Posición X, Y, Z, tiempo, satélites a la vista, HDOP.
- **Mensaje 4. Measure Tracker Data Out.** Datos de la constelación de satélites, incluyendo PRN, azimuth, elevación, C/N0 para cada uno de los 10 slots del correlador.
- **Mensaje 27. DGPS Status Format.** Datos sobre la fuente de datos DGPS.
- **Mensaje 28. Navigation Library Measurement Data.** Observables de navegación, incluyendo tiempo GPS, pseudorrango, y fase de la portadora.
- **Mensaje 41. Geodetic Navigation Data.** Tiempo, latitud, longitud, altitud sobre el elipsoide, altitud sobre el nivel del mar, velocidad, orientación, variación de la altura, de la

orientación, errores HPE, VPE, error de reloj, de orientación, HDOP...

- **Mensaje 50. SBAS Parameters.** Datos del satélite de la constelación SBAS –sistema de corrección basado en satélites geoestacionarios, que incluye la europea EGNOS, la americana WAAS, etc.– en uso.

5.3 Proveedor de localización a través de protocolo TSIP

El protocolo TSIP (Trimble Standard Interface Protocol) fue **desarrollado por Trimble para la comunicación con dispositivos GPS que utilizan su chipset**. Al igual que SiRF, se trata de un protocolo binario con una velocidad binaria elevada. Trimble se dedica a la fabricación de GPS de alta gama, por lo que el protocolo **permite el intercambio de gran cantidad de datos avanzados y de especial interés para usuarios profesionales del ámbito de la geodesia**.

Estos datos incluyen observables del GPS –pseudorrango, fase de portadora, dopler, etc.– que pueden ser almacenados en ficheros **RINEX** para corrección diferencial en postproceso, y potencialmente pueden ser usados también para corrección diferencial en tiempo real.

En estos momentos disponemos de una implementación básica que sólo decodifica un mensaje TSIP:

- **Mensaje 0x84. Double-Precision LLA Position Fix & Clock Bias.** Latitud, longitud, altitud, tiempo y desfase del reloj.

5.4 Proveedor de localización a través de demonio GPSd

GPSd es un **demonio de Linux** –una aplicación corriendo en segundo plano– **que permite compartir sin conflictos un dispositivo GPS entre varias aplicaciones**. Para ello se crea un socket en la máquina conectada al dispositivo en el que se ofrece un stream de datos en formato texto plano. A este socket pueden conectarse aplicaciones en la misma máquina en la que está conectado el GPS y también en máquinas remotas.

GPSd da soporte a dispositivos GPS operando con un amplio rango de protocolos, incluyendo los protocolos NMEA 0183, SiRF, TSIP, Garmin, EverMore, Navcom, Rockwell/Zodiac, y uBlox ANTARIS, por lo que **es una opción para conectar con dispositivos no soportados directamente por libLocation**.

6 Trabajo futuro

El primer objetivo es **completar la integración de la librería en la versión 1.0 de gvSIG Mobile** mediante el desarrollo de varias extensiones.

Posteriormente y a medio plazo se trabajaría en las siguientes mejoras sobre la librería:

- Autoconfiguración de proveedores de localización.
- Desarrollo de un gestor de las comunicaciones que permita controlar y asignar de forma flexible los input streams y output streams a diferentes clases de la aplicación mediante listeners. Esto permitirá mayor flexibilidad a la hora de conectarse por ejemplo a un servidor NTRIP, sin necesidad de configurarlo antes de abrir el LocationProvider.
- Aplicación de correcciones diferenciales por software en tiempo real (DGPS y/o RTK).
- Almacenamiento de datos basado en DAL (librería de acceso a datos para gvSIG 2.0).