6. Interfaces del programa (Georap).

6.1 Introducción

Se puede definir de forma general que una interfaz es un sistema hardware y software que conecta dos aparatos ó dispositivos. Para el programa GEORAP es necesario que haya dos tipos de interfaces, una encargada de obtener los datos de posición del GPS y otra para recibir las correcciones diferenciales para que el GPS funcione como DGPS.

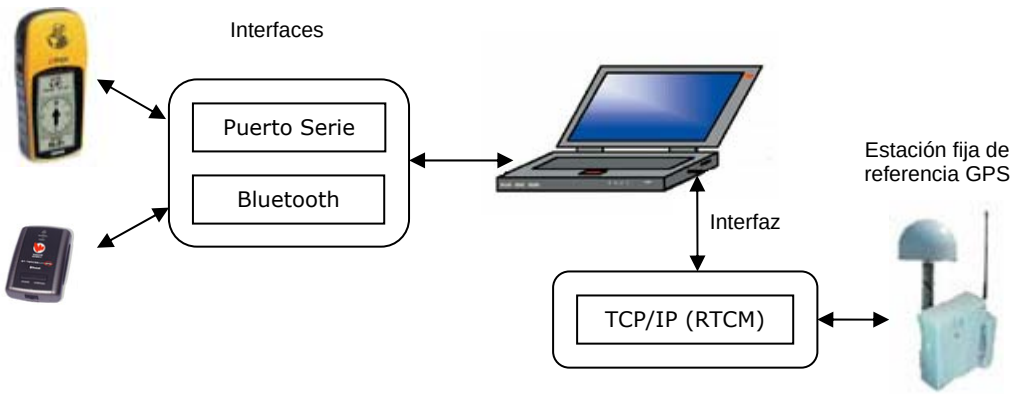

Figura 6.1. Interfaces del sistema

6.2. Comunicación con el equipo GPS

6.1.1. Puerto Serie (RS232)

Este programa proporciona la opción de comunicación con el dispositivo GPS a través de algún puerto serie del ordenador. Para que la transmisión de los datos se realice satisfactoriamente es necesario configurar y poner los mismos parámetros de comunicación en ambos aparatos, por defecto aparece: puerto serie (COM1), velocidad de transferencia (4800 baudios), sin control de flujo de entrada ni de salida, bits de datos (8), bits de stop (1) y sin bits de paridad. Como aparece en la figura 6.2.

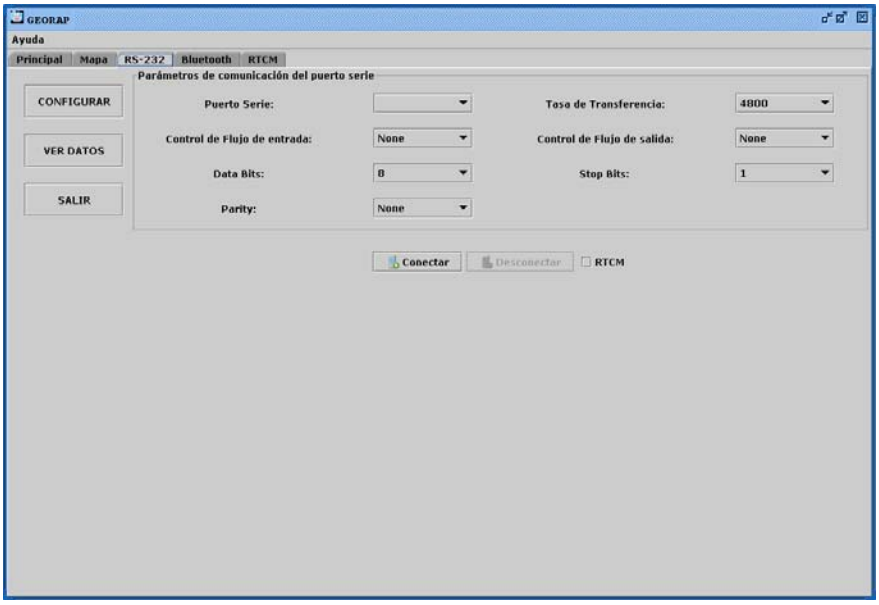

Figura 6.2. Imagen de configuración RS232

Implementación en JAVA

Se ha utilizado la API CommAPI (Java™ *Communications API*) para controlar los puertos serie con la aplicación Java. Todas las clases empleadas se han almacenado en el paquete RS232 de la aplicación.

Clases empleadas:

- **RS232.java**

Proporciona un panel para la interfaz RS232, donde permite cambiar de subpanel central tras la pulsación de los botones "CONFIGURAR" ó "VER DATOS".

- **Serial.java**

Facilita un panel para poder configurar los parámetros de la comunicación serie, a través de elementos ComboBox. Esta clase también permite establecer la conexión por el puerto serie y la habilitación la transmisión de la información RTCM usando un CheckBox.

- **ParámetrosSerie.java**

Almacena los valores de los parámetros de la transmisión serie, donde pueden ser cambiados o leídos.

- **ConexionSerie.java**

La clase ConexionSerie es la encargada de realizar y supervisar la conexión RS232, e implementa `SerialPortEventListener` para escuchar los eventos producidos en el puerto serie.

Los métodos que la componen son:

- **abrirConexion()**: Intenta abrir una conexión serie, con sus flujos de datos de entrada y salida, usando los parámetros de configuración de la comunicación serie. Si algún paso no es soportado por el puerto se lanza una excepción y se cierra.
- **poneParametros()**: Pone los parámetros de la conexión. Si ocurre algún fallo cuando se ponen los parámetros en el puerto serie se deja el valor original de los parámetros.
- **cerrarConexion()**: Cierra el puerto y borra los elementos asociados.
- **serialEvent(SerialPortEvent e)**: Método que escucha los eventos producidos por el puerto serie. Diferencia entre los dos tipos de eventos utilizados (`DATA_AVAILABLE` y `Break_Interrupt`). Cuando ocurre un evento `DATA_AVAILABLE` se lee el buffer del puerto hasta que no hay más datos y pasan 30 ms hasta pasar la respuesta del método. Cuando ocurre un evento de ruptura de transmisión es indicado en la caja de texto de entrada, y no lee datos.
- **enviarPuerto(byte[ ] b, int off, int len)**: Se encarga de enviar los bytes de longitud "len" introducidos como parámetro al puerto serie, empezando por el byte correspondiente a la posición "off".

- **EnviarPuerto(int dato)**: Se encarga de enviar el entero pasado como parámetro al puerto serie.

- **VerDatos.java**

Visualiza los datos transmitidos tanto de entrada como de salida del puerto serie.

6.1.2. Bluetooth

a) Nociones Básicas:

Bluetooth es un estándar de facto global que identifica un conjunto de protocolos que facilitan la comunicación inalámbrica entre diferentes tipos de dispositivos electrónicos. El nombre procede del rey danés y noruego Harald Blåtand cuya traducción al inglés sería *Harold Bluetooth* (Diente Azul, aunque en lengua danesa significa 'de tez oscura') conocido por unificar las tribus noruegas, suecas y danesas.

Bluetooth es el nombre común de la especificación industrial IEEE 802.15.1, que define un estándar global de comunicación inalámbrica que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia segura, globalmente y sin licencia de corto rango. Los principales objetivos que se pretende conseguir con esta norma son:

1. Facilitar las comunicaciones entre equipos móviles y fijos.
2. Eliminar cables y conectores entre éstos.
3. Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre nuestros equipos personales.

Bluetooth es una tecnología de comunicación inalámbrica omnidireccional y opera en la banda libre de radio ISM (banda internacional médico-científica) en el rango de 2,4 a 2,48 GHz. La especificación de Bluetooth define un canal de comunicación de máximo 720 kb/s (1 Mbps de capacidad bruta) con rango óptimo de 10 metros (opcionalmente 100 m con repetidores).

La potencia de salida para transmitir a una distancia máxima de 10 metros es de 0 dBm (1 mW), mientras que la versión de largo alcance transmite entre 20 y 30 dBm (entre 100 mW y 1 W).

Para lograr alcanzar el objetivo de bajo consumo y bajo costo, se ideó una solución que se puede implementar en un solo chip utilizando circuitos CMOS (*Complementary Metal Oxide Semiconductor)*. De esta manera, se logró crear una solución de 9x9 mm y que consume aproximadamente 97% menos energía que un teléfono celular común.

- Salto de frecuencia**:**

Debido a que la banda ISM está abierta a cualquiera, el sistema de radio Bluetooth deberá estar preparado para evitar las múltiples interferencias que se pudieran producir. Éstas pueden ser evitadas utilizando un sistema que busque una parte no utilizada del espectro o un sistema de salto de frecuencia.

En este caso la técnica de salto de frecuencia es aplicada a una alta velocidad y una corta longitud de los paquetes (1600 saltos/segundo). Con este sistema se divide la banda de frecuencia en varios canales de salto, donde, los transceptores, durante la conexión, van cambiando de uno a otro canal de salto de manera pseudo-aleatoria.

Los paquetes de datos están protegidos por un esquema ARQ (repetición automática de consulta), en el cual los paquetes perdidos son automáticamente retransmitidos.

- El canal**:**

Bluetooth utiliza un sistema FH/TDD (salto de frecuencia/división de tiempo duplex), en el que el canal queda dividido en intervalos de 625 µs, llamados *slots*, donde cada salto de frecuencia es ocupado por un *slot*.

Dos o más unidades Bluetooth pueden compartir el mismo canal dentro de una *piconet* (pequeña red que establecen automáticamente los terminales Bluetooth para comunicarse entre si), donde una unidad actúa como maestra, controlando el tráfico de datos en la *piconet* que se genera entre las demás unidades, donde éstas actúan como esclavas, enviando y recibiendo señales hacia el maestro.

El salto de frecuencia del canal está determinado por la secuencia de la señal, es decir, el orden en que llegan los saltos y por la fase de esta secuencia. En

Bluetooth, la secuencia queda fijada por la identidad de la unidad maestra de la *piconet* (un código único para cada equipo), y por su frecuencia de reloj.

- Arquitectura Hardware

El hardware que compone el dispositivo Bluetooth esta compuesto por dos partes:

1. **un dispositivo de radio**, encargado de modular y transmitir la señal
2. **un controlador digital**, compuesto por una CPU, por un procesador de señales digitales (DSP - *Digital Signal Processor*) llamado *Link Controller* (o controlador de Enlace) y de los interfaces con el dispositivo anfitrión.

- Comunicación Bluetooth:

En una comunicación Bluetooth existe un dispositivo que ofrece un servicio (servidor) y otros dispositivos acceden a él (clientes). Dependiendo de qué parte de la comunicación debamos programar deberemos realizar una serie de acciones diferentes.

Un cliente Bluetooth deberá realizar las siguientes acciones:

1. Búsqueda de dispositivos. La aplicación realizará una búsqueda de los dispositivos Bluetooth a su alcance que estén en modo conectable.
2. Búsqueda de servicios. La aplicación realizará una búsqueda de servicios por cada dispositivo.
3. Establecimiento de la conexión. Una vez encontrado un dispositivo que ofrece el servicio deseado nos conectaremos a él.
4. Comunicación. Ya establecida la conexión podremos leer y escribir en ella.

Por otro lado, un servidor Bluetooth deberá hacer las siguientes operaciones:ç

1. Crear una conexión servidora
2. Especificar los atributos de servicio
3. Abrir las conexiones clientes

La aplicación creada, hace de cliente y el dispositivo GPS se comporta como servidor. Por tanto, el código implementado efectúa todas las funciones descritas.

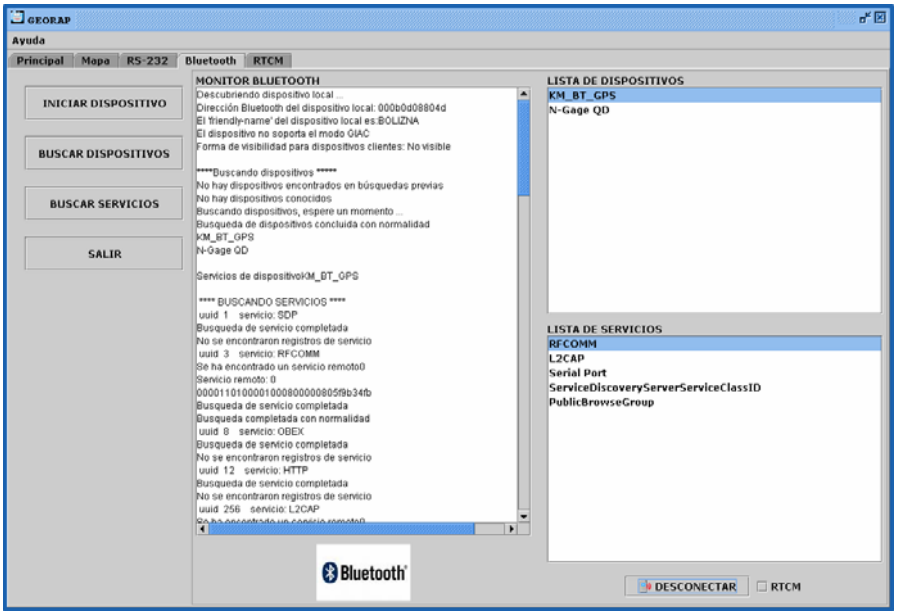

Figura 6.3. Panel Bluetooth

b) **Implementación en JAVA**[1]

La API JSR 82 estandarizó la forma de desarrollar aplicaciones Bluetooth usando JAVA, así se conseguía esconder la complejidad del protocolo Bluetooth detrás de unos APIs que permiten centrarse en el desarrollo en vez de los detalles de bajo nivel del Bluetooth.

Este API está dividida en dos partes: el paquete javax.bluetooth y el paquete javax.obex. Los dos paquetes son totalmente independientes. El primero de ellos define clases e interfaces básicas para el descubrimiento de dispositivos, descubrimiento de servicios, conexión y comunicación. La comunicación a través de javax.bluetooth es a bajo nivel: mediante flujos de datos o mediante la transmisión de arrays de bytes.

Por el contrario el paquete javax.obex permite manejar el protocolo de alto nivel OBEX (OBject EXchange). Este protocolo es muy similar a HTTP y es utilizado sobre todo para el intercambio de archivos. El protocolo OBEX es un estándar desarrollado por IrDA y es utilizado también sobre otras tecnologías inalámbricas distintas de Bluetooth. En esta versión del programa no ha sido utilizado dado que para la comunicación con el dispositivo GPS no ha sido necesario, pero no se

[1] Para obtener más información sobre la implementación de la interfaz bluetooth se puede dirigir al código fuente de la aplicación, y a la documentación de la bibliografía.

descarta su uso en futuras versiones, dado que con su utilización se puede dotar al programa de un sistema de transferencia de ficheros entre ordenadores, móviles, o a otros dispositivos.

La plataforma principal de desarrollo del API JSR-82 es J2ME. La API ha sido diseñada para depender de la configuración CLDC. Sin embargo, existen implementaciones para poder hacer uso de este API en J2SE. En las siguientes tablas podemos observar kits de desarrollo para utilizar este API.

- Kits completos de desarrollo (incluyen el hardware verdadero de Bluetooth no simulado, el software de Java, los manuales, y código del ejemplo).

Nombre del producto	Ayuda de javax.bluetooth	Ayuda de javax.obex	Java Plataformas	Sistemas operativos
BlipNet	No	No	J2SE	Win-32, Linux
JB-22	Sí	Sí	J2SE	Win-32
Possio Px 30	Sí	No	J2ME	Win-32, Linux

Tabla 6.1. Kits de desarrollo completos

- Kits de desarrollo con sólo software:

Nombre de compañía	ayuda de javax.bluetooth	ayuda de javax.obex	Plataformas Java	Funcionamiento Sistemas	OpenSource
Atinav	Sí	Sí	J2ME, J2SE	Win-32, Linux, PC del bolsillo	No
Avetana	Sí	Sí	J2SE	Win-32, OS X, Linux, PC Mac de bolsillo	No, sólo para Linux
Bluecove	Sí	No	J2SE	WinXP SP2	Si
Harald	No	No	cualquier plataforma que apoye **javax.comm**	muchos	Si
JavaBluetooth.org	Sí	No	cualquier plataforma que apoye **javax.comm**	muchos	Si
Rococo	Sí	Sí	J2ME, J2SE	Linux, OS de la palma	No

Tabla 6.2. Kits de desarrollo sólo software

Se ha optado por la utilización de BlueCove dado que es una implementación *Open Source* del JSR-82 que utiliza la implementación del API JSR 82 de Bluetooth para Windows XP. Además, en la página del proyecto BlueCove afirman que desean soportar otros sistemas operativos a posteriori.

Dentro del paquete Bluetooth, se encuentran las clases construidas para implementar esta interfaz en el programa y que son descritas a continuación:

- **Bluetooth.java**

Esta encargada de crear el panel para la interfaz gráfica con el usuario y de realizar las tareas necesarias para la conexión bluetooth con el GPS.

- **inicio():** se llama cuando se pulsa el botón "INICIAR DISPOSITIVO" y se dedica a descubrir el dispositivo local, obteniendo el objeto de representación del dispositivo local que pertenece a la clase "LocalDevice".
- **buscar_dispostivos():** entra en funcionamiento tras pulsar "BUSCAR DISPOSITIVOS". El primer paso es comprobar si existen dispositivos encontrados en búsquedas anteriores, ó anteriormente definidos como conocidos con el usuario. Las búsquedas de dispositivos y servicios Bluetooth se realizan a través del objeto `DiscoveryAgent`. Para comenzar una nueva búsqueda de dispositivos llamaremos al método `startInquiry()`. Este método requiere dos argumentos. El primer argumento es un entero que especifica el modo de conectividad que deben tener los dispositivos a buscar. Este valor deberá ser `DiscoveryAgent.GIAC` (conectividad ilimitada) o bien `DiscoveryAgent.LIAC` (conectividad limitada), en nuestra aplicación hemos utilizado el modo GIAC. El segundo argumento es un objeto que implemente `DiscoveryListener`. A través de este último objeto serán notificados los dispositivos que se vayan descubriendo.
- **buscarServicios(RemoteDevice dispositivo_remoto):** Es utilizada para obtener los servicios asociados a un dispositivo. Para realizar una búsqueda de servicios también usaremos la clase `DiscoveryAgent`, y su método `searchServices`(). Este método es un poco complejo por lo se verá con más detalle:
  public int `searchServices(int[] attrSet, UUID[] uuidSet, RemoteDevice btDev, DiscoveryListener discListener) throws BluetoothStateException`; El primer argumento es un array de enteros con el que especificaremos los atributos de servicio en los que estamos

interesados. Pues bien, este array contendrá los identificadores de estos atributos. El segundo argumento es un array de identificadores de servicio. Nos permite especificar los servicios en los que estamos interesados. El tercer argumento es el dispositivo remoto sobre el que vamos a realizar la búsqueda. Por último argumento pasaremos un objeto que implemente `DiscoveryListener` que será usado para notificar los eventos de búsqueda de servicios.

- **conectar():** efectúa la conexión con el dispositivo y a través del servicio seleccionado, para ello se utilizan la clase `ServiceRecord`, y con su método `getConnectionURL`, se obtiene la URL asociada a este servicio en el dispositivo remoto. Con la URL obtenida podemos obtener un objeto `StreamConnection`, y desde él se puede abrir los flujos de entrada y salida de datos para la conexión bluetooth. Por último, se crea el objeto del tipo `Conexión` para supervisar la conexión, y leer los datos que se reciben del GPS.
- **enviaRTCM(char dato):** Este método realiza la función de transmitir la información RTCM obtenida vía internet al dispositivo GPS Bluetooth, cuando se activa la caja de selección RTCM.

Subclases:

- **Listener:** implementa la interfaz DiscoveryListener y para su buen uso es necesario crear los siguientes métodos:

  - **deviceDiscovered(RemoteDevice dispositivoRemoto, DeviceClass clase):** Cada vez que se descubre un dispositivo se llama a este método. Nos pasa dos argumentos. El primero es un objeto de la clase `RemoteDevice` que representa el dispositivo encontrado. El segundo argumento nos permitirá determinar el tipo de dispositivo encontrado.

  - **inquiryCompleted(int discType):** Este método es llamado cuando la búsqueda de dispositivos ha finalizado. Nos pasa un argumento entero indicando el motivo de la finalización (búsqueda concluida con normalidad, error en la búsqueda o búsqueda cancelada).

  - **servicesDiscovered(int transID, ServiceRecord[ ] servRecord):** Este método nos notifica que se han encontrado servicios. El primer argumento es un entero que identifica el proceso de búsqueda.

El segundo argumento es un array de objetos `ServiceRecord`. Un objeto ServiceRecord describe las características de un servicio bluetooth. Un servicio Bluetooth se describe mediante atributos, los cuales se identifican numéricamente, es decir, un servicio Bluetooth tiene una lista de pares identificador-valor que lo describen. El objeto `ServiceRecord` se encarga de almacenar estos pares.

- **serviceSearchCompleted(int transID, int respCode):** Este método es llamado cuando se finaliza un proceso de búsqueda. El primer argumento identifica el proceso de búsqueda. El segundo argumento indica el motivo de finalización de la búsqueda (concluida con normalidad, cancelada, no se encontraron registros, error en la búsqueda, o no se pudo conectar al dispositivo sobre el que se pretende hacer la búsqueda de servicios).

- **CheckBoxListener:** implementa la interfaz ItemListener y se encarga de escuchar el evento producido al seleccionar ó deseleccionar la opción RTCM, que se encarga en transmitir los datos RTCM al dispositivo GPS.

- **Conexión.java**

Se crea el objeto del tipo `Thread` para supervisar la conexión bluetooth, y leer los datos que se reciben del GPS, todo ello con la conexión activa.

Se ha optado por la creación de este proceso para no sobrecargar al proceso general del programa.

6.1.3. NMEA

NMEA (*National Marine Electronics Association*) es una asociación sin fines de lucro de fabricantes, distribuidores, instituciones educacionales y otros interesados en equipos periféricos marinos. El estándar NMEA define una interfaz eléctrica y un protocolo de datos para la comunicación entre instrumentos marinos y, también, más generalmente, receptores GPS. La salida NMEA es EIA-422A, pero para la mayoría de los propósitos puede considerarlo RS-232 compatible.

Este protocolo se lanza por primera vez en marzo de 1983, y su última versión fue publicada en el 2001.

Bajo el estándar NMEA-0183 todos los caracteres usados son escritos en texto ASCII (además del retorno de carro). Cada sentencia comienza con un "$", dos letras como "talker ID", tres letras como "sentencia ID" seguidas de una serie de campos de datos separados por comas, y termina con un checksum opcional y un retorno de carro (CR). La sentencia debe contener como máximo 82 caracteres. Si un campo de datos se omite por no estar disponible o por cualquier motivo, las comas que lo delimitan deben enviarse sin espacio entre ellas, para que así el receptor cuente las comas e interprete correctamente cada campo. El campo checksum es opcional, se identifica porque comienza con "*", y está formado por dos dígitos hexadecimales que representan la XOR de todos los caracteres de la sentencia entre "$" y "*" (ambos excluidos).

Figura 6.4. Sentencia NMEA

Algunos "talker ID":

- GP: receptor GPS.
- LC: receptor Loran-C.
- OM: receptor de navegación Omega.
- II: Instrumentación Integrada.

Las "sentencias ID" utilizadas por el programa son:

- **GGA (Información de posicionamiento)**

$GPGGA, 161229.487,3723.2475,N,12158.3416,W,1,07,1.0,9.0,M,,,,.0000*18

Nombre	Ejemplo	Unidades	Descripción
Identificador del mensaje	$GPGGA		Cabecera del mensaje
Hora UTC	161229.487		hhmmss.ss
Latitud	3723.2475		ggmm.mmmm
Indicador N/S	N		N = norte ó S = Sur
Longitud	12158.3416		gggmm.mmmm
Indicador E/W	W		E = Este ó W = Oeste
Modo de Funcionamiento del dispositivo GPS	1		0 ==> No válido 1 ==> Simple GPS 2 ==> Diferencial GPS 3 ==> Preciso GPS 4 ==> Tiempo real cinemático 5 ==> Flotante Tiempo Real Cinemático 6 ==> Estimado 7 ==> Entrada manual 8 ==> Simulación
Satélites usados			Rango entre 0 a 12
HDOP	1.0		Dilución Horizontal de la Posición
MSL Altitud	9.0	Metros	Altitud de la Antena Sobre/Bajo Nivel del Mar Intermedio (geoide)
Unidades	M	Metros	
Separación geoidal		Metros	Separación Geoidal (Diferencia entre elipsoide y nivel del mar intermedio, si es negativo el geoide está bajo el elipsoide)
Unidades	M	Metros	
Tiempo del la última corrección diferencial		Segundos	Intervalo en segundos desde la última actualización de una estación de referencia diferencial.
Identificador de la estación diferencial	0000		Identificador de la estación de referencia diferencial.
Checksum	*18		Suma de comprobación
<CR><LF>			Fin del mensaje

Tabla 6.3. Mensaje GGA

- **GLL (Datos de latitud y longitud)**

$GPGLL, 3723.2475,N,12158.3416,W,161229.487,A*2C

Nombre	Ejemplo	Unidades	Descripción
Identificador del mensaje	$GPGLL		Cabecera del mensaje
Latitud	3723.2475		ggmm.mmmm
Indicador N/S	N		N = norte ó S = Sur
Longitud	12158.3416		gggmm.mmmm
Indicador E/W	W		E = Este ó W = Oeste
Hora UTC	161229.487		hhmmss.ss
Estado	A		A ==> Válido V ==> No válido
Checksum	*2C		Suma de comprobación
<CR><LF>			Fin del mensaje

Tabla 6.4. Mensaje GLL

- **GSA (Lista de satélites usados en la solución)**

$GPGSA,A,3,01,20,19,13,,,,,,,,,1.8,1.0,1.5*0A

Nombre	Ejemplo	Unidades	Descripción
Identificador del mensaje	$GPGSA		Cabecera del mensaje
Estado	A		A ==> Válido V ==> No válido
Modo	3		1=No Fijado, 2=2D,3=3D
Canal 1 (PRN satélite)	01		Satélite en Canal 1
Canal 2	20		Satélite en Canal 2
.....	.		...
Canal 12			Satélite en Canal 12
PDOP	1.8		Precisión de Posición Dilución
HDOP	1.0		Precisión de Dilución Horizontal
VDOP	1.5		Precisión de Dilución Vertical
Checksum	*0A		Suma de comprobación
<CR><LF>			Fin del mensaje

Tabla 6.5. Mensaje GSA

- **GSV (GNSS Satélites en vista)**

$GPGSV,2,1,07,79,048,42,02,51,062,43,26,36,256,42,27,27,138,42*71

Nombre	Ejemplo	Unidades	Descripción
Identificador del mensaje	$GPGSV		Cabecera del mensaje
Nº de mensajes	2		Rango 1 a 3
Mensaje	1		
Satélites en vista	07		
Id. Satélite	07		Canal 1 (1~32)
Elevation	79	Grados	Canal 1 (max. 90)
Azimuth	048	Grados	Canal 1 0-359
SNR (C/No)	42	dB	SNR, 00-99 dB (nulo cuando no hay track)
...			...
Satellite ID	27		Channel 4 (1~32)
Elevation	27	Grados	Canal 4 (max. 90)
Azimuth	138	Grados	Canal 4 0-359
SNR (C/No)	42	dB	SNR, 00-99 dB (nulo cuando no hay track)
Checksum	*71		Suma de comprobación
<CR><LF>			Fin del mensaje

Tabla 6.6. Mensaje GGA

- **RMC (Recomendación Mínima Específica de datos GNSS)**

$GPRMC,161229,A,3723.2475,N,12158.3416,W,005.6,123.5,020197,001.0,W,A
07CRLF

Nombre	Ejemplo	Unidades	Descripción
Identificador del mensaje	$GPGGA		Cabecera del mensaje
Hora UTC	161229		hhmmss (16:hh 12:mm 29:ss) Rango: hh=00-23 mm=00-59 ss=00-59
Estado	A		A ==> Válido V ==> No válido
Latitud	3723.2475		ggmm.mmmm
Indicador N/S	N		N = norte ó S = Sur
Longitud	12158.3416		gggmm.mmmm
Indicador E/W	W		E = Este ó W = Oeste
Velocidad			
			Rango entre 0 a 12
Fecha: UTC	1.0		Dilución Horizontal de la Posición
Variación Magnética	001.0 W	grados	Rango = 000.0-180.0 W o E. E = Este(MAG=TRU+DEV) o W = Oeste(MAG=TRU-DEV)
Indicador de modo de sistema de posicionamiento			A: Modo automático D: Modo diferencial N: Datos no válidos
Checksum	*18		Suma de comprobación
<CR><LF>			Fin del mensaje

Tabla 6.7. Mensaje RMC

- **VTG (Norte y velocidad)**

$GPVTG, 309.62, T, , M, 0, 13, N, 0, 2, K*6E

Nombre	Ejemplo	Unidades	Descripción
Identificador del mensaje	$GPVTG		Cabecera del mensaje
Rumbo	309.62	grados	Grados aproximados
Referencia	T		Norte Verdadero
Rumbo		grados	Grados aproximados
Referencia	M		Norte Magnético
Velocidad	0.13	nudos	Velocidad horizontal aproximada
Unidades	N		Nudos (1 nudo = 1.85 km/h)
Velocidad	0.2	km/h	Velocidad horizontal aproximada
Unidades	K		Rango entre 0 a 12
Checksum	*6E		Suma de comprobación
<CR><LF>			Fin del mensaje

Tabla 6.8. Mensaje VTG

- **ZDA (Fecha y hora)**

$GPZDA, 123456, 01, 02, 1997, +09,00, *6B

Nombre	Ejemplo	Unidades	Descripción
Identificador del mensaje	$GPZDA		Cabecera del mensaje
Hora UTC	123456		hhmmss (12h 34m 56s) Rango: hh=00-23 mm=00-59 ss=00-59
UTC: día del mes	01	dd	01~31
UTC: mes	02	mm	01~12
UTC: año	1997	yyyy	1997~2040
Zona Horaria local (horas)	+09 hh	-13...+00...+13	
Zona Horaria local (minutos)	00	mm	00~59
Checksum	*6B		Suma de comprobación
<CR><LF>			Fin del mensaje

Tabla 6.9. Mensaje ZDA

Nota: Hora local = UTC – Zona horaria local

Implementación en JAVA

Clases:

- NMEA.java

Se ha creado esta clase para tratar las líneas de datos recibidas de equipo GPS en formato NMEA. Haciendo una distinción de los campos de cada mensaje NMEA, obteniendo el tipo de cada secuencia.

- ponerdatos(String): recoge en el objeto NMEA una nueva línea para obtener sus campos.
- tipo(): obtiene el tipo de secuencia NMEA que hay almacenada.
- GGA(String): Crea un vector con los campos del mensaje GGA.
- GSA(String): Crea un vector con los campos del mensaje GSA.
- RMC(String): Crea un vector con los campos del mensaje RMC.
- GSV(String): Crea un vector con los campos del mensaje GSV.
- ZDA(String): Crea un vector con los campos del mensaje ZDA.
- VTG(String): Crea un vector con los campos del mensaje VTG.

6.2. Comunicación con las estaciones de referencia

6.2.1. Introducción

Se ha dotado al sistema de dos modos de comunicación con las estaciones fijas GPS, el primero de ellos es el modo automático y el segundo es el modo manual. A través de ellos el programa se conecta a la estación de referencia y obtiene la información en el formato RTCM-SC104.

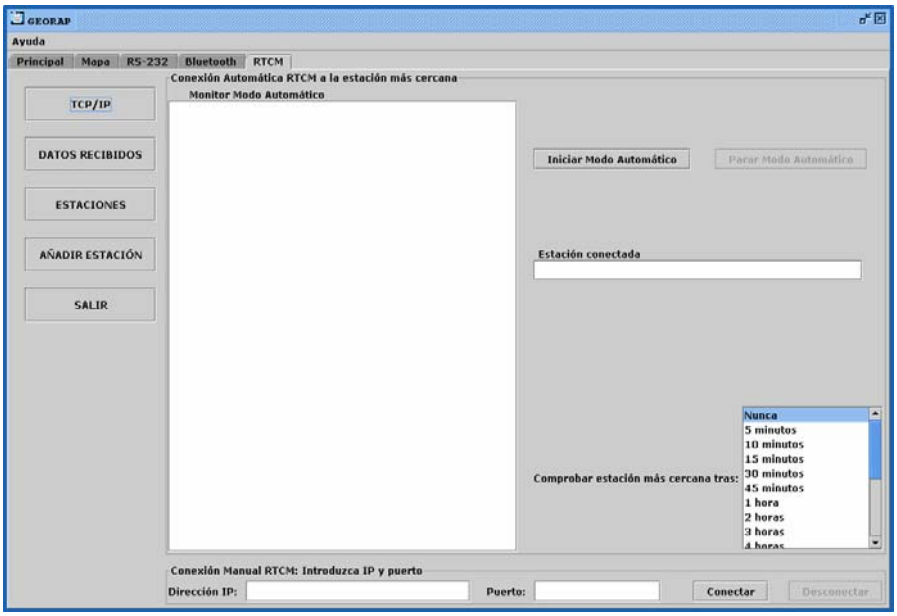

Figura 6.5. Panel RTCM

6.2.2 Formatos de envío de correcciones en tiempo real.

Los mensajes de corrección RTCM (*Radio Technical Commission for Maritime Service*) son el formato estándar para la utilización de los sistemas diferenciales de posicionamiento. Las decisiones a cerca de los criterios y de la manera en que han de implementarse estos mensajes son tomadas por un grupo perteneciente a la organización RTCM, se trata del Sub – Comité 104. Existen diferentes versiones de este formato, las cuales se han ido adaptando a las necesidades que el sistema demandaba en cada momento.

En 1983 la Radio *Technical Commission for Maritime Service* fundó el Comité Especial 104 con la finalidad de desarrollar las recomendaciones y criterios para la transmisión de correcciones de pseudodistancias (entonces sólo para usuarios de código). Esto se llevó a cabo 1985 con la primera versión RTCM 1.0. En 1998 se publicó la versión 2.2, en 2001 la versión 2.3 y en el 2004 salió la última versión 3.0.

Actualmente la mayoría de las estaciones envían sus mensajes en formato RTCM 2.3, aunque se está incrementando el número de estaciones que está actualizándose transmitiendo en la versión 3.0. Algunos sistemas también aceptan correcciones transmitidas en formato CMR (*Compact Measurement Record - Registro compacto de medición*) y CMR+, de la empresa Trimble, y en formato Leica de la empresa del mismo nombre.

También existe un protocolo de nueva creación para transportar las correcciones RTCM vía internet, llamado NTRIP (*Networked Transport of RTCM via Internet Protocol*). Este protocolo ha sido desarrollado por el BKG junto con la universidad de Dortmund y está basado en http (*Hyper Text Transfer Protocol*), permite transmitir el flujo DGPS así como cualquier otro tipo de datos GNSS a receptores móviles mediante Internet.

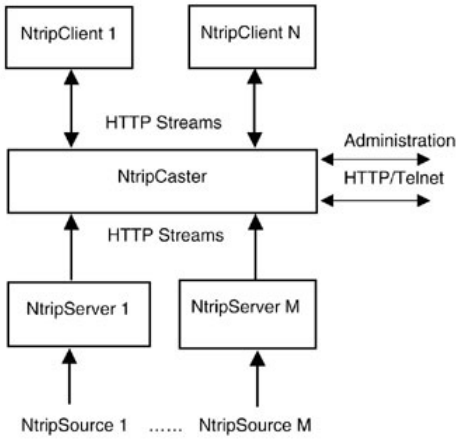

Figura 6.6. Componentes de NTRIP

Los componentes de NTRIP son:

- NtripSources: que generan flujos de datos DGPS en una localización determinada.
- NtripServers: que transfieren los datos desde una o varias fuentes en formato NTRIP.
- NtripCaster: corresponde al sistema de difusión y repartidor de flujos.
- NtripClients: los receptores de datos de las distintas fuentes desde el caster.

Las ventajas de esta tecnología son:

- El protocolo NTRIP está basado en http como capa de protocolo en la parte más alta de TCP/IP (modelo OSI). Consecuentemente cualquier flujo de datos es transmitido exclusivamente a través del puerto 80, de esta manera los problemas habituales con los *firewall* y *proxies* se eliminan.

- El *caster* actúa entre el cliente y el servidor de forma similar a la radio por internet, duplica los flujos de datos entrantes de tal manera que puedan ser recibidos simultáneamente por varios usuarios. Éste paso intermedio actúa a la vez como elemento de seguridad para los proveedores de datos.

6.2.1. Modo automático

Esta forma de realizar la comunicación permite la conexión a la estación más cercana de la posición del dispositivo y además que este operativa. Es necesario tener el GPS en funcionamiento para poder calcular la estación más cercana calculando la distancia de su posición a las estaciones.

Se puede configurar el programa para que vaya comprobando la estación más cercana y operativa tras un periodo de tiempo (nunca, 5 minutos, 10 minutos, así aumentando progresivamente hasta un máximo de un día). Este servicio es adecuado para situaciones donde utilizando el GPS se desplace moderada ó rápidamente, y además, para comprobar la operatividad de las estaciones.

Se dispone de la opción "AÑADIR ESTACIÓN" para ir cargando estaciones fijas GPS en el programa, y podemos ver las estaciones introducidas pulsando el botón "ESTACIONES", que por defecto aparecerán las estaciones de la RAP (Red Andaluza de Posicionamiento). También cabe la posibilidad de eliminar estaciones que vayan sido mal introducidas ó hayan sido sustituidas.

6.2.2. Modo manual

Para utilizar el modo manual es necesario saber la dirección IP y puerto de la estación de referencia. Una vez introducidas sólo hay que pulsar en conectar, y si le estación está en funcionamiento, se recibirán las correcciones RTCM.

6.2.3. Implementación software[2]

Todas las clases construidas para realizar esta interfaz han sido almacenadas en el paquete RTCM de la aplicación.

- RTCM.java

Se encarga de crear el panel principal para la comunicación con las estaciones de referencia, a este panel se le adjuntan todos los demás.

[2] Para obtener más detalles de la implementación software de la comunicación con las estaciones se puede dirigir al paquete RTCM del programa, hay encontrará el código fuente comentado de todas las clases.

- TCPIP.java

Crea el panel TCP/IP, y a través de éste podemos conectarnos de forma manual ó automáticamente para obtener los datos RTCM. Nos conectaremos de modo manual introduciendo la IP más el puerto y seleccionando "Conectar", en ese momento el programa obtiene la IP con el puerto y crea un `Socket` (conexión a través del Protocolo Internet (IP)) a esa estación, y engendra un objeto hilo del tipo `ComTcpIp` para leer el flujo de datos recibido. Para terminar se pulsará "Desconectar", que hace que se finalice el proceso `ComTcpIp` y se cierre el `Socket`.

Desde esta clase también se inicia el modo automático, y se ha implementando básicamente creando un objeto y activándolo `AutoTcpIp` que es un proceso encargado en el funcionamiento del modo automático. Para parar el modo automático simplemente finalizaremos este proceso.

- AutoTcpIp.java

Esta clase es más complicada de entender, se ha optado por crear una clase del tipo Thread, es decir, por crear un proceso ó hilo, para efectuar automáticamente la conexión y comprobar periódicamente la estación más próxima. Se mostrará su funcionamiento por pasos:

1. Comprueba si hay conexión con el GPS.
2. Obtiene la estación más cercana con el método minimo();
3. Crea un `Socket` a esa estación, y engendra un objeto hilo del tipo `ComTcpIp` para leer el flujo de datos recibido.
4. Si la conexión no se puede realizar se pone la estación como no válida y se intenta con la siguiente más cercana, así sucesivamente, hasta conectarse con alguna o no haber más estaciones para comprobar.
5. Espera el tiempo prefijado y realiza los pasos anteriores.

- ComTcpIp.java

Esta clase es del tipo Thread y se utiliza para leer los datos que se reciben de las estaciones de referencia. Este proceso hace que el proceso principal no se sobrecargue, dado que mientras este la conexión en funcionamiento, se tiene que hacer cargo de leer continuamente el flujo de datos.

- Estaciones.java

Se usa la clase JTable para crear una tabla donde cada fila representa los datos de una estación (IP, puerto, latitud, longitud, altitud, coordenada cartesiana X, coordenada cartesiana Y, coordenada cartesiana Z y el nombre de la estación).

- SumarEstacion.java

Desde aquí podemos añadir estaciones al programa, es necesario indicar como mínimo la dirección IP y el puerto de la nueva estación, cuando no se indique ningún dato de posición se tomarán como latitud 0º0´0”N, longitud 0º0´0”E y altitud 0 metros. También se da la posibilidad de añadir la nueva estación en la lista de estaciones predeterminadas.

- DatosRTCM.java

Esta clase proporciona un panel donde se muestra los datos brutos que se reciben de las estaciones.