

<b>Título</b>	TTT_Servidor_Documentación_Final		
<b>Proyecto</b>	Práctica de la asignatura Proyectos Informáticos: Tiempos del Transporte de Teruel (TTT)		
<b>Equipo</b>	<p>Jorge Navarrete Argilés</p> <p>Jesús Fuentes Romero</p> <p>Óscar Munárriz Sesma</p> <p>Ángel Sánchez Espílez</p>		
<b>Código</b>	TTT Serv Docu	<b>Fecha</b>	16/06/11
<b>Profesor</b>	Félix Serna Fortea	<b>Versión</b>	1.0

## 0. ÍNDICE

<b>0. ÍNDICE.....</b>	<b>2</b>
<b>1. INTRODUCCION.....</b>	<b>4</b>
1.1 Propósito.....	4
1.2 Alcance.....	4
1.3 Definiciones, acrónimos y abreviaturas.....	5
<b>2. DESCRIPCIÓN GLOBAL.....</b>	<b>6</b>
2.1 Perspectiva del producto.....	6
2.2.1 Interfaz de sistema.....	6
2.1.2 Interfaz de usuario.....	7
2.1.3 Interfaz software.....	7
2.1.4 Diagramas de Casos de Uso.....	8
<b>3. CARACTERÍSTICAS DEL PRODUCTO.....</b>	<b>9</b>
3.1 Características generales.....	9
3.2 Características del protocolo.....	9
3.3 Restricciones de sistema.....	9
<b>4. REQUISITOS ESPECÍFICOS.....</b>	<b>10</b>
4.1 Requisitos generales.....	10
4.2 Requisitos del protocolo.....	10
4.2.1 getParadas.....	10
4.2.2 Refrescar tiempos.....	10
4.2.3 getAutobuses.....	10
4.2.4 getLineas.....	10
4.2.5 Establecer info autobús.....	11
4.2.6 Refrescar info autobús.....	11
4.3 Requisitos de sistema.....	12
<b>5. DIAGRAMAS UML.....</b>	<b>13</b>
5.1 Diagramas de clases.....	13
5.2 Diagramas de actividad.....	17

<b>5.3 Diagramas de secuencia.....</b>	<b>18</b>
--	-----------

# 1. INTRODUCCION

El presente documento contiene una Especificación de Requisitos de Software (ERS) para la aplicación informática de acceso a datos del servidor. Esta es la parte central del sistema de “Tiempos de Transporte de Teruel”, a realizar durante la asignatura Proyectos Informáticos. La estructura del presente documento está inspirada en las directrices referidas en el estándar IEEE Recommended Practice for Software Requirements Specifications IEEE 830-1998.

El objetivo principal de esta aplicación es proporcionar a las aplicaciones cliente los datos que necesiten para funcionar, así como recibir las actualizaciones de la información de los autobuses.

## 1.1 Propósito

El objeto de la especificación de requisitos es definir de manera clara y precisa tanto las funcionalidades como las posibles restricciones del sistema de información a construir. Dicho documento está dirigido tanto a los miembros del equipo de desarrollo como al cliente del proyecto. En él, se establecen las bases sobre las cuales el equipo de desarrollo procederá al diseño y posterior construcción del nuevo sistema. También servirá de modelo en el cual basar las pruebas funcionales de aceptación por parte del cliente y la evaluación final del producto por parte del mismo.

## 1.2 Alcance

El producto a obtener será una aplicación que proporcione conectividad entre la base de datos y las aplicaciones cliente, para responder con los datos requeridos a las peticiones que se le realice. Además deberá atender las actualizaciones de la información de los autobuses.

### 1.3 Definiciones, acrónimos y abreviaturas

<b>Línea</b>	Secuencia de paradas donde un viajero puede coger un autobús para llegar a otra de las paradas de la línea
<b>Sentido</b>	Cada una de las direcciones en las que se puede recorrer una línea
<b>Ruta</b>	El trazado que une las paradas de una línea
<b>Parada</b>	Punto localizado geográficamente donde los autobuses de ciertas líneas pueden dejar o coger viajeros
<b>Servidor</b>	Aplicación no orientada a los usuarios cuya misión es servir a las aplicaciones cliente
<b>Cliente</b>	Aplicación que necesita información del servidor
<b>Petición</b>	Solicitud de una aplicación cliente en la que espera una respuesta del servidor

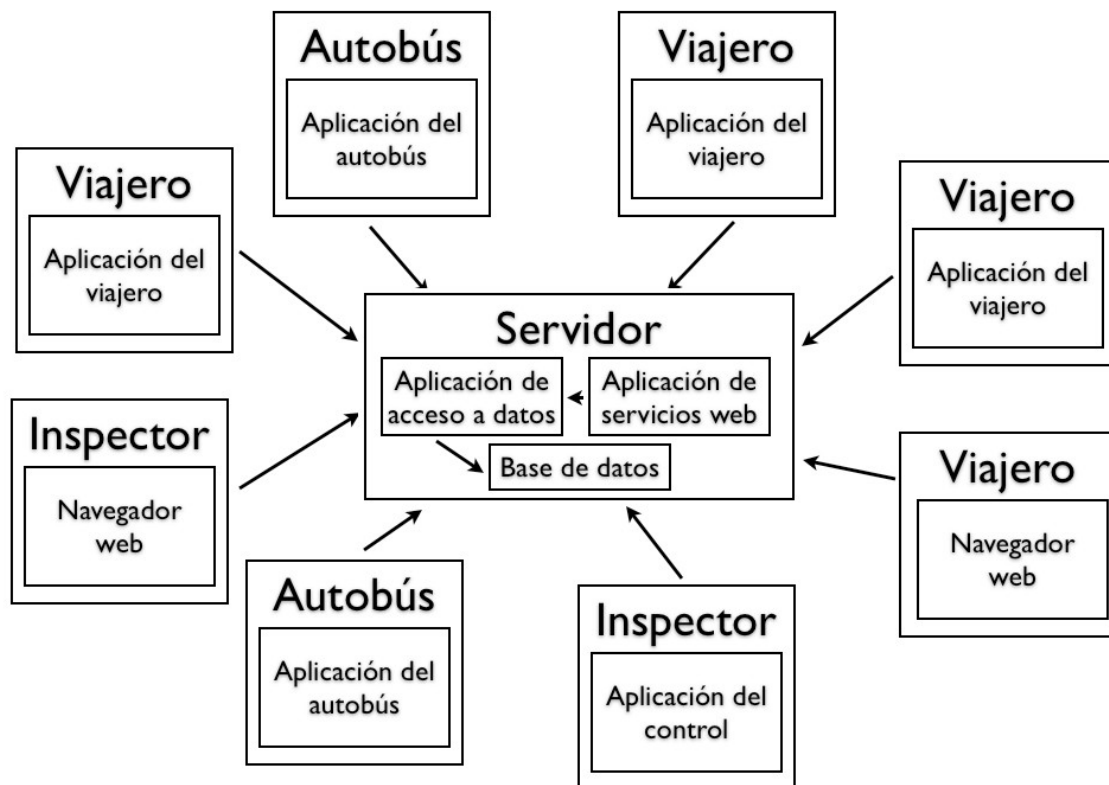
## 2. DESCRIPCIÓN GLOBAL

### 2.1 Perspectiva del producto

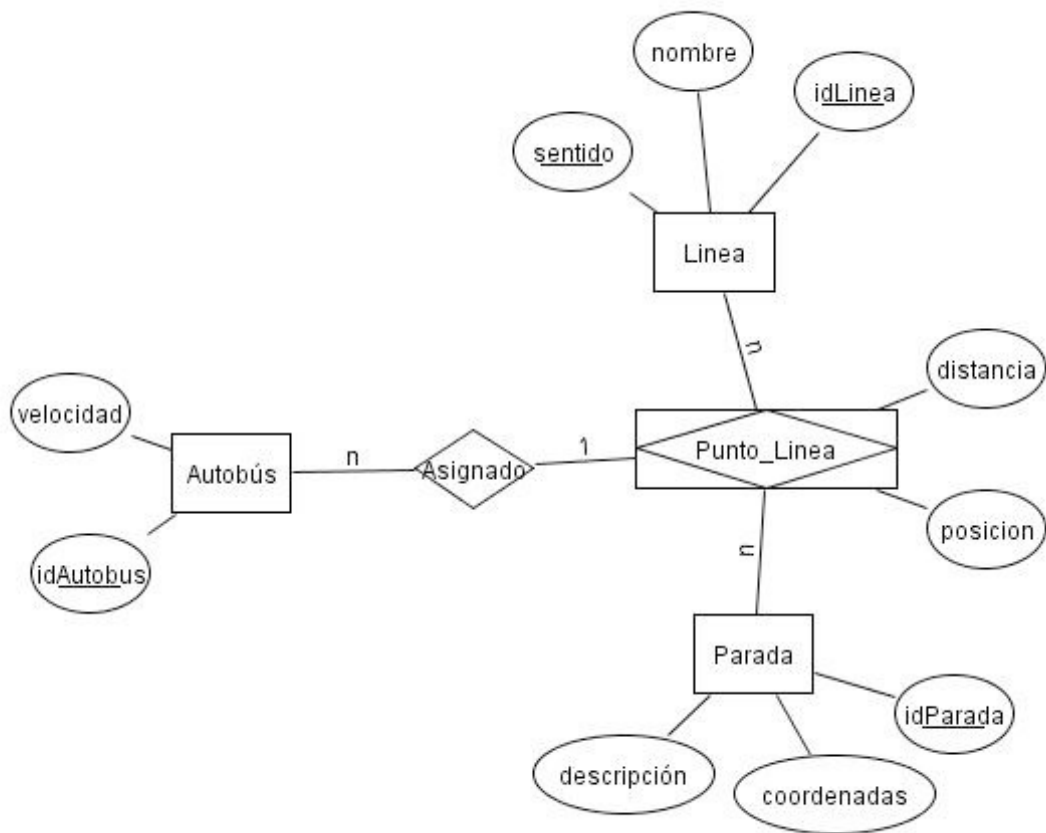
El sistema necesita de una aplicación central que ofrezca datos actualizados a las aplicaciones cliente, y donde los autobuses puedan conectarse para refrescar sus datos. Para ello se debe seguir el protocolo establecido en el apartado 4 de la documentación común.

#### 2.2.1 Interfaz de sistema

La aplicación del servidor de datos es la parte central del sistema, a la que todas las aplicaciones cliente se conectan.



A la vez, esta aplicación necesita acceso a una base de datos de donde obtener la información y almacenarla. En este caso va a tratarse de una base de datos MySQL siguiendo el siguiente esquema entidad-relación:



### 2.1.2 Interfaz de usuario

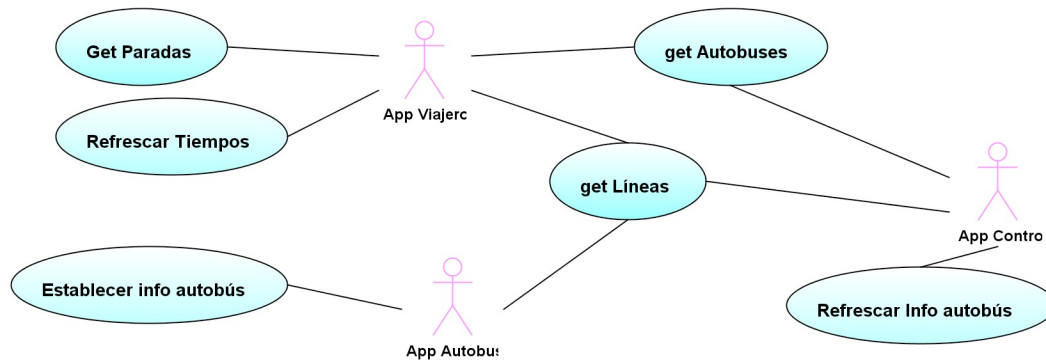
Esta aplicación no dispone de interfaz de usuario.

### 2.1.3 Interfaz software

Se requiere JRE 1.6 instalado.

Además se necesita conectividad con una base de datos MySQL que puede estar instalada en el mismo servidor.

### 2.1.4 Diagramas de Casos de Uso



**Get Paradas:** La aplicación cliente hace una solicitud para que se le devuelva un listado de todas las paradas del sistema, con su información de ubicación geográfica, descripción de la parada y número de parada.

**Refrescar Tiempos:** La aplicación cliente hace una solicitud con un número de parada y el servidor debe responder con un listado de líneas-sentido, descripción de la línea y el tiempo en minutos que falta para que llegue el siguiente autobús.

**getAutobuses:** La aplicación cliente hace una solicitud para obtener del servidor un listado de autobuses con el número de autobús, ubicación geográfica y línea-sentido al que está asignado.

**getLineas:** La aplicación cliente hace una solicitud para obtener del servidor un listado de líneas-sentido y sus descripciones.

**Establecer info autobús:** La aplicación del autobús envía una solicitud para actualizar sus datos con su número de autobús, número de parada que ha alcanzado (o bien vacío al empezar), velocidad media, línea y sentido al que está asignado. El servidor debe responder con la siguiente parada a la que debe ir el autobús, dando la distancia que hay, descripción de la parada, número de la parada y ubicación geográfica.

**Refrescar info autobús:** La aplicación de control hace una solicitud con un número de autobús para obtener la información del autobús: Línea-sentido, velocidad media, la descripción de la última parada y su ubicación geográfica.



## 3. CARACTERÍSTICAS DEL PRODUCTO

### 3.1 Características generales

**CAR\_GEN\_01:** El sistema deberá estar siempre activo.

**CAR\_GEN\_02:** El sistema deberá estar siempre conectado a Internet.

**CAR\_GEN\_03:** Se deben responder las peticiones de las aplicaciones cliente.

**CAR\_GEN\_04:** Cada petición debe ocupar el sistema el mínimo tiempo posible, liberando los recursos al haber devuelto la respuesta.

### 3.2 Características del protocolo

**CAR\_PRO\_01:** La aplicación debe responder a las peticiones que se reflejan en los casos de uso.

**CAR\_PRO\_02:** Cuando se reciba una petición se debe construir la respuesta con los datos más actualizados. Si la petición lo requiere la aplicación realizará los cambios pertinentes en los datos.

**CAR\_PRO\_03:** Todas las peticiones deben tener una respuesta de acuerdo al protocolo establecido en el apartado 4 de la documentación común.

### 3.3 Restricciones de sistema

**RES\_GEN\_01:** Se requiere JRE 1.6.

**RES\_GEN\_02:** Debe haber un servidor conectado mediante un enlace rápido (puede ser el mismo) que aloje la base de datos sobre MySQL.

**RES\_GEN\_03:** Las aplicaciones cliente establecerán una conexión con la aplicación del servidor solo durante el intercambio de datos. No se mantendrá la misma conexión para más de un intercambio.

## 4. REQUISITOS ESPECÍFICOS

### 4.1 Requisitos generales

**REQ\_GEN\_01\_01:** El servidor almacena la información en base de datos y permite el intercambio de información con los clientes.

**REQ\_GEN\_02\_01:** Solo la aplicación cliente del Autobús tiene permitido actualizar la información del sistema. El resto de aplicaciones cliente se limitan a la consulta.

**REQ\_GEN\_02\_02:** Las aplicaciones obtienen la información cuando la solicitan.

### 4.2 Requisitos del protocolo

#### 4.2.1 *getParadas*

**REQ\_PRO\_01\_01:** La aplicación puede recibir una solicitud de una aplicación cliente pidiendo un listado de paradas.

**REQ\_PRO\_01\_02:** La aplicación debe responder al cliente que ha realizado la solicitud con un listado de todas las paradas del sistema, indicando para cada una su número de parada, descripción y ubicación geográfica.

#### 4.2.2 *Refrescar tiempos*

**REQ\_PRO\_02\_01:** La aplicación puede recibir una solicitud de una aplicación cliente pidiendo los tiempos de espera de una parada identificada por su número de parada.

**REQ\_PRO\_02\_02:** La aplicación debe calcular en ese momento el tiempo estimado de llegada del autobús más cercano que no haya pasado ya por la parada.

**REQ\_PRO\_02\_03:** La aplicación debe responder al cliente que ha realizado la solicitud con un listado de todas las líneas y sentidos con su descripción indicando el tiempo estimado de llegada en minutos.

#### 4.2.3 *getAutobuses*

**REQ\_PRO\_03\_01:** La aplicación puede recibir una solicitud de una aplicación cliente pidiendo un listado de los autobuses que están en servicio, es decir, cubriendo alguna línea.

**REQ\_PRO\_03\_02:** La aplicación debe responder al cliente que ha realizado la solicitud con un listado de todos los autobuses asignados a alguna línea, indicando para cada una su número de autobús, línea-sentido y ubicación geográfica.

#### 4.2.4 *getLineas*

**REQ\_PRO\_04\_01:** La aplicación puede recibir una solicitud de una aplicación cliente pidiendo un listado de las líneas de la ciudad.

**REQ\_PRO\_04\_02:** La aplicación debe responder al cliente que ha realizado la solicitud con un listado de todas las líneas-sentido del sistema, indicando para cada una su línea, sentido y nombre.

#### ***4.2.5 Establecer info autobús***

**REQ\_PRO\_05\_01:** La aplicación puede recibir una solicitud de una aplicación cliente de un autobús indicando sus nuevos datos en el momento que empieza a recorrer una línea o que alcanza una parada.

**REQ\_PRO\_05\_02:** La aplicación debe actualizar en ese momento la nueva información en la base de datos y calcular la siguiente parada que le corresponde al autobús.

**REQ\_PRO\_05\_03:** La aplicación cliente acompaña la solicitud con la información del número de autobús, línea-sentido asignado, velocidad media y número de parada (si no acaba de comenzar la línea).

**REQ\_PRO\_05\_04:** La aplicación debe responder a la solicitud con el número de la siguiente parada, su descripción, su ubicación geográfica y la distancia con la parada anterior, es decir, el recorrido hasta la siguiente parada.

#### ***4.2.6 Refrescar info autobús***

**REQ\_PRO\_06\_01:** La aplicación puede recibir una solicitud de una aplicación cliente pidiendo información de un autobús identificado por su número de autobús.

**REQ\_PRO\_06\_02:** La aplicación debe responder al cliente que ha realizado la solicitud con la velocidad media entre las dos paradas anteriores del autobús consultado; el número, la descripción y la ubicación geográfica de la última parada alcanzada.

### ***4.3 Requisitos de sistema***

**REQ\_SIS\_01:** La aplicación del servidor deberá atender las peticiones realizadas por las aplicaciones cliente.

**REQ\_SIS\_02:** Se puede atender a un número de clientes no limitado.

**REQ\_SIS\_03:** La conexión se realiza siguiendo un protocolo por establecer.

**REQ\_SIS\_05:** Las aplicaciones cliente deben mantener la conexión con el servidor el mínimo tiempo posible.

**REQ\_SIS\_07:** La base de datos debe estar construida sobre MySQL siguiendo el esquema entidad-relación indicado en la documentación.

**REQ\_SIS\_08:** La base de datos debe almacenar todos los datos necesarios para poder atender las peticiones necesarias para satisfacer las peticiones de los clientes.

## 5. DIAGRAMAS UML

### 5.1 Diagramas de clases

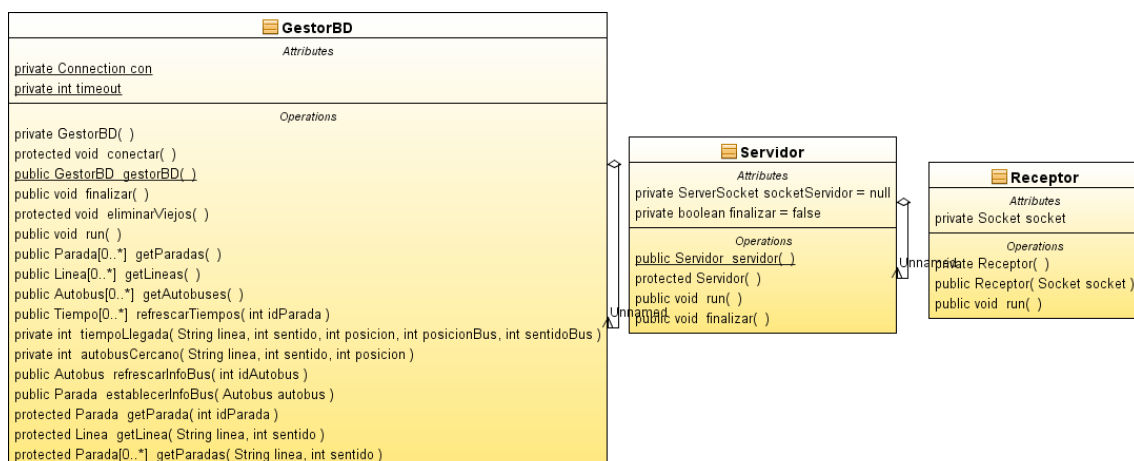


La clase Linea contiene la información del identificador de la línea, el sentido de la marcha en el que se circula, y el nombre de la línea. Tiene dos métodos para comprobar cuál es la siguiente parada a la que se debe ir (siguienteParada()) y el otro para agregar más paradas a las pertenecientes a esa línea (agregarParada()). El resto de métodos son get y set además de los constructores.

Por otra parte la clase Tiempo se encarga de almacenar el tiempo de paso de un autobús por una parada y sirve para poder procesar las peticiones del servidor. Pide las diferentes peticiones necesarias para la gestión del viajero, como obtener todas las paradas de una determinada línea. Esta clase solo contiene métodos get y set además de los constructores.

La clase Parada por su parte almacena todos los datos de una parada como su identificador, su descripción su posición y la distancia a la anterior parada. Esta clase sólo contiene métodos get y set junto con sus constructores.

La clase Autobus almacena el identificador del autobús, la velocidad actual, la línea que está recorriendo así como la última parada por la que ha pasado.



Las clases GestorBD, Servidor y Receptor son las que modelan el funcionamiento del servidor de datos. Además, se requieren otras clases que serán las que se creen para enviar a los clientes con la información solicitada.

### GestorBD

Es una clase de la que solo puede existir una instancia (Singleton). Su cometido es conectar a la base de datos y realizar las consultas requeridas. Sus métodos públicos son:

**gestorBD():** Devuelve la única instancia de la clase. Si no existe, además conectará con la base de datos indicada en el fichero etc/basedatos.xml y ejecutará un hilo que se encarga de eliminar de la base de datos aquellos autobuses que lleven demasiado tiempo sin realizar modificaciones mediante llamadas a los métodos constructor y conectar().

**finalizar():** Desconecta de la base de datos. Se deberá usar solo antes de finalizar la aplicación.

**run():** Este método es llamado automáticamente, no se debe utilizar desde el exterior. Su misión es ejecutarse indefinidamente y periódicamente para eliminar los autobuses que hayan estado demasiado tiempo sin actualizar datos.

**getParadas():** Consulta en la base de datos todas las paradas y las devuelve en un listado de objetos de clase Parada con la descripción de éstas, ubicación e identificador.

**getLineas():** Consulta en la base de datos todas las líneas y las devuelve en un listado de objetos de clase Linea con el nombre, el identificador de línea y el sentido. Además incluye un listado ordenado de las paradas por las que pasa con la descripción de éstas, ubicación, identificador y distancia en metros desde la anterior usando el método getParadas(línea,sentido).

**getAutobuses():** Consulta en la base de datos todos los autobuses que estén recorriendo alguna línea y los devuelve en un listado de objetos de clase Autobus con el número de autobús, la velocidad media, la línea que está recorriendo de la misma forma que hace getLineas() con cada línea, y la última parada por la que ha pasado con su identificador, descripción, ubicación y distancia con la anterior.

**refrescarTiempos(idParada):** Devuelve el listado de Tiempo que representan los tiempos de espera en segundos del siguiente autobús en pasar de cada línea en una parada determinada. Para conseguirlo, para cada línea que pasa por la parada, busca el autobús más cercano usando el método autobusCercano(línea,sentido,posición), donde posición es el punto de la parada dentro de la línea. Si no hubiera un autobús por llegar, miraría si lo hay recorriendo el otro sentido para el tiempo total hasta llegar a la parada usando el método tiempoLlegada, que suma las distancias y lo divide por la velocidad del autobús.

**refrescarInfoBus(idAutobus):** Consulta en la base de datos para obtener la información del autobús con el número indicado. Finalmente devuelve una instancia de la clase Autobus con la información del número de autobús, velocidad media, la línea que está recorriendo y la última parada que ha realizado.

**establecerInfoBus(autobus):** Es el único método que introduce información en la base de datos. A partir de la instancia de clase Autobus del parámetro, extrae la información de la línea y sentido, última parada realizada, la velocidad media y el número de autobús para insertar esta información junto a una marca de tiempo en la base de datos o, si el número de autobús ya existe, actualizar la existente. Devuelve la siguiente parada a la que se debe dirigir el autobús.

## **Servidor**

Es la clase encargada de aceptar las conexiones entrantes. Sus métodos públicos son:

**servidor():** Se trata de una clase con una sola instancia, con lo que este método devuelve la instancia. Si no existe la crea. Al crear la clase, lee el puerto por el que llegarán las conexiones vía socket desde el fichero etc/servidor.xml.

**run():** Para iniciar el servidor, un método main debe crear un thread que invoque este método, ya que es bloqueante. Mientras no se llame al método **finalizar()**, espera y acepta conexiones en un bucle, creando un thread de la clase Receptor por cada conexión.

## **Receptor**

Es la clase encargada de gestionar la conexión con un cliente, recibiendo sus mensajes y enviando la respuesta. Sus métodos públicos son:

**Receptor(Socket):** Una vez aceptada una conexión, se llama al constructor para crear una instancia que se dedique a este cliente. Inmediatamente se debe crear un Thread de esta instancia y ejecutarlo.

**run():** Es el método que se ejecuta cuando se inicia el hilo. Es el encargado de abrir los canales de comunicación por el socket, leer el tipo de mensaje (Clase enumerada TipoPetición) y seguidamente leer el objeto con la posible especificación de información. A continuación llama al método correspondiente de GestorBD para construir la respuesta y enviarla al cliente.

Dependiendo del tipo de petición, se llamará al método de GestorBD que corresponda:

getParadas -> getParadas()

getLineas -> getLineas()

getAutobuses -> getAutobuses()


refrescarTiempos -> refrescarTiempos(idParada)

refrescarInfoBus -> refrescarInfoBus(idAutobus)

establecerInfoBus -> establecerInfoBus(autobus)

En los tres últimos casos, el Receptor habrá leído un objeto después

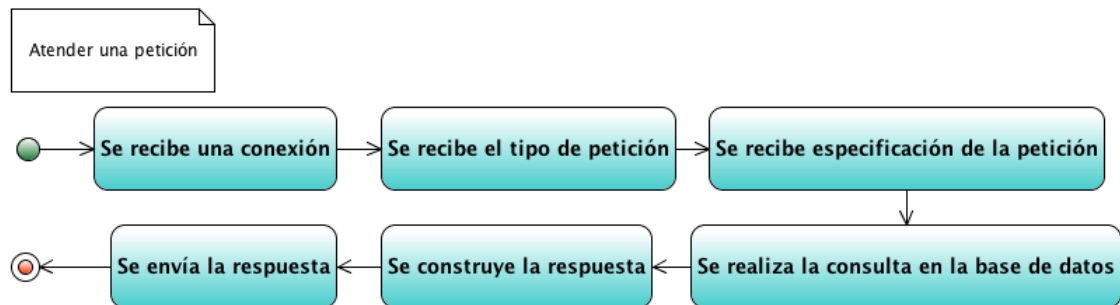
de leer el tipo de petición. En los tres primeros, esta información no se utiliza.

 TipoPetición
<i>Literals</i>
getParadas
getLineas
getAutobuses
refrescarTiempos
refrescarInfoBus
establecerInfoBus

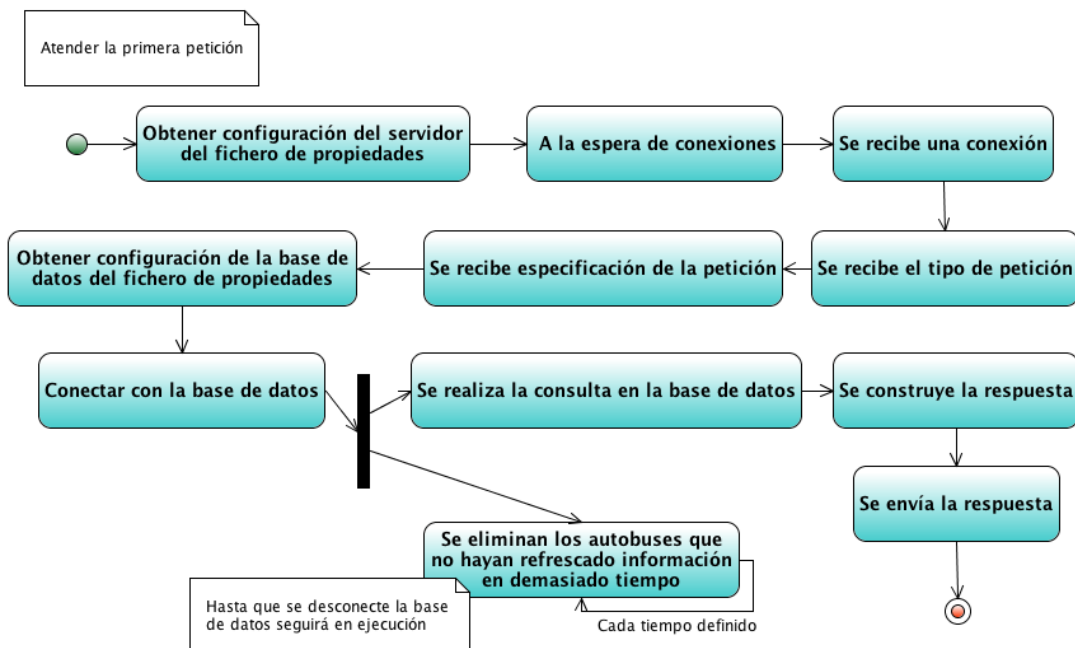
## 5.2 Diagramas de actividad

La misión de la aplicación es devolver la información solicitada a partir de los datos encontrados en la base de datos. De esa forma, el funcionamiento es homogéneo para los seis casos de uso, aunque debido a las acciones que se realizan al generar las instancias únicas de las clases se puede diferenciar una secuencia de actividades mayor en la inicialización.

Representamos cualquiera de las seis consultas de la siguiente manera:



El proceso donde se inicializan las instancias de Servidor y de GestorBD se representa:





### 5.3 Diagramas de secuencia

Se indica el diagrama de secuencia correspondiente a una consulta del tipo refrescarTiempos en una parada con idParada. El resto de peticiones se consideran similares. Además se muestra en el diagrama las inicializaciones de las instancias, como se puede apreciar en las llamadas desde el hilo “main” y el método run() que GestorBD se invoca:

