ANEXO A. DISEÑO DETALLADO

1. PLATAFORMA DE DESARROLLO

Básicamente el objetivo general del proyecto podría resumirse en tan sólo el concepto correspondiente a la creación de una arquitectura abierta modular que permita el desarrollo e implementación de servicios basados en el posicionamiento. En las siguientes páginas se ilustra los casos de uso, sus diagramas de secuencia y el respectivo diagrama de clases que permitieron llegar a la consecución de esa meta.

El anexo será dividido en dos partes, la primera es la correspondiente al análisis de la Plataforma de Desarrollo y la segunda hace referencia a los Servicios implementados.

1.1 Diagrama de Casos de Uso

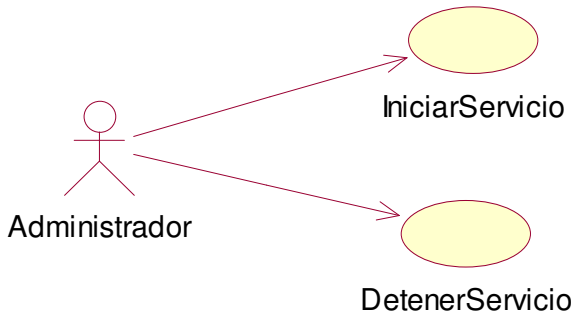

Figura 1. Diagrama de Casos de Uso (Administrador)

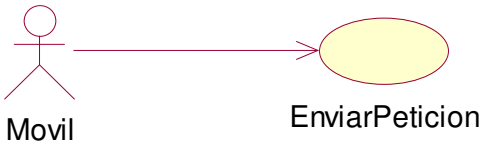

Figura 2. Diagrama de Casos de Uso (Móvil)

1.2 Descripción de casos de Uso

- **Caso de uso No. 1:** *Iniciar Servicio*

  Iniciador: Administrador

  Precondición: El servicio debe existir

  Propósito: Permitir que el servicio se encuentre activo para que el sistema pueda procesar las peticiones.

  Resumen: Este caso de uso empieza cuando el administrador mediante una interfaz activa el servicio en el servidor de aplicaciones.

  Flujo principal:

  1. El administrador o prestador de servicio presiona el botón que activa el servicio.
  2. El sistema genera la red[1], inicia el rastreo de parlay, se obtiene el framework, el gestor de servicios, se crea el procesador y se comienza la notificación SMS/MMS.
  3. El sistema inicia a esperar peticiones del servicio.

  Flujos alternos:

  - Ninguno

[1] Red: hace referencia a la red de carreteras o vías usada por donde se desplaza el usuario, en este caso corresponde a la zona del centro de Popayán.

- **Diagrama de Secuencia**

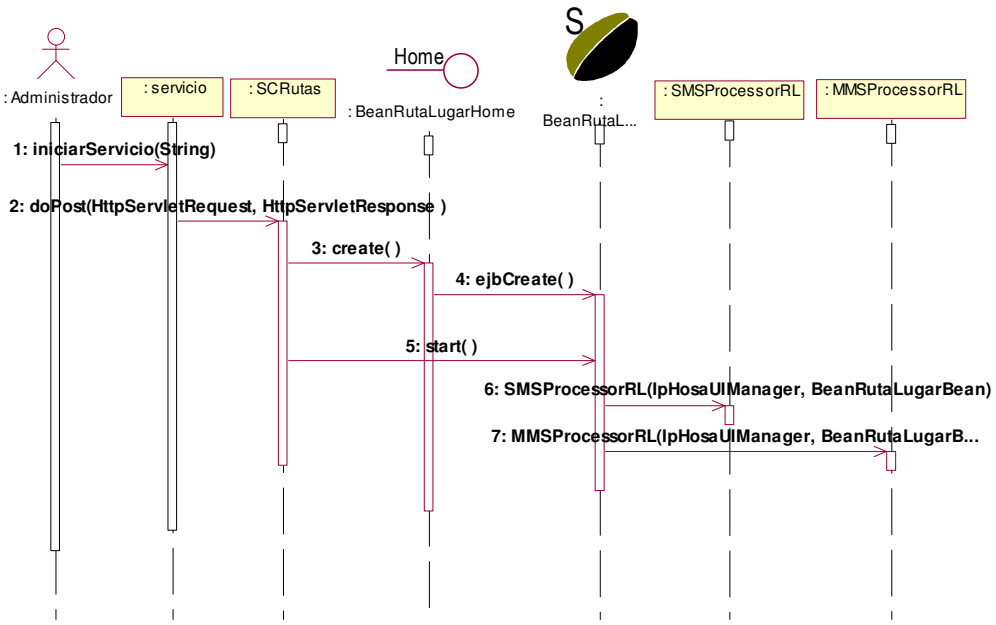

Figura 3. Diagrama de Secuencia Iniciar Servicio

Este diagrama muestra la secuencia seguida por el sistema cuando el administrador o prestador de servicios inicia o pone en funcionamiento el servicio.

El administrador accede a la interfaz gráfica representada por la clase Servicio donde tiene la opción de iniciar o detener el servicio, el sistema es capaz de reconocer si el servicio ya ha sido iniciado por lo que deshabilita el botón de inicio para evitar confusiones y errores. Al seleccionar el botón de inicio de la interfaz gráfica, se llama al método doPost de la clase SCRutas que en este caso es la clase de control del servicio al que se le esta dando inicio, esta clase crea el objeto del EJB correspondiente al servicio y lo pone a correr, cuando se llama al método start del EJB se crean los objetos de las clases que permiten el procesamiento de los mensajes de texto y multimedia; a partir de este momento se habilita la prestación del servicio y se pueden recibir las peticiones de los usuarios.

- **Caso de uso No. 2:** *Enviar petición*

  Iniciador: Terminal Móvil

  Precondición: El servicio debe estar en funcionamiento

  Propósito: Permitir el envío de la información correspondiente para que sea procesada por el sistema.

  Resumen: Este caso de uso empieza cuando el usuario del móvil mediante un mensaje de texto proporciona su ubicación dentro de la zona de prueba (centro de la ciudad de Popayán), junto con los demás datos necesarios para el servicio que desea usar y envía este mensaje al código del servicio.

  Flujo principal:

  1. El usuario del dispositivo móvil envía un mensaje con la información necesaria al número correspondiente al servicio.
  2. El sistema responde analizando la información enviada y generando un mensaje de texto o multimedia según el servicio y enviándolo como respuesta al terminal.

  Flujos alternos:

  - El número al que se envía el mensaje de petición es incorrecto.
    2. El mensaje no es enviado porque el servicio no existe o se trata de procesar la información enviada en el servicio al que corresponde el número del mensaje por lo que pasaría al siguiente flujo alterno.
  - La información que se envía al servicio es incorrecta o incompleta
    2. El sistema envía un mensaje de texto informándole al usuario que alguno de los datos es incorrecto.
  - El servicio no ha sido iniciado o no existe.
    1. El mensaje no puede ser enviado. (aparece como mensaje no enviado)

- **Diagrama de Secuencia**

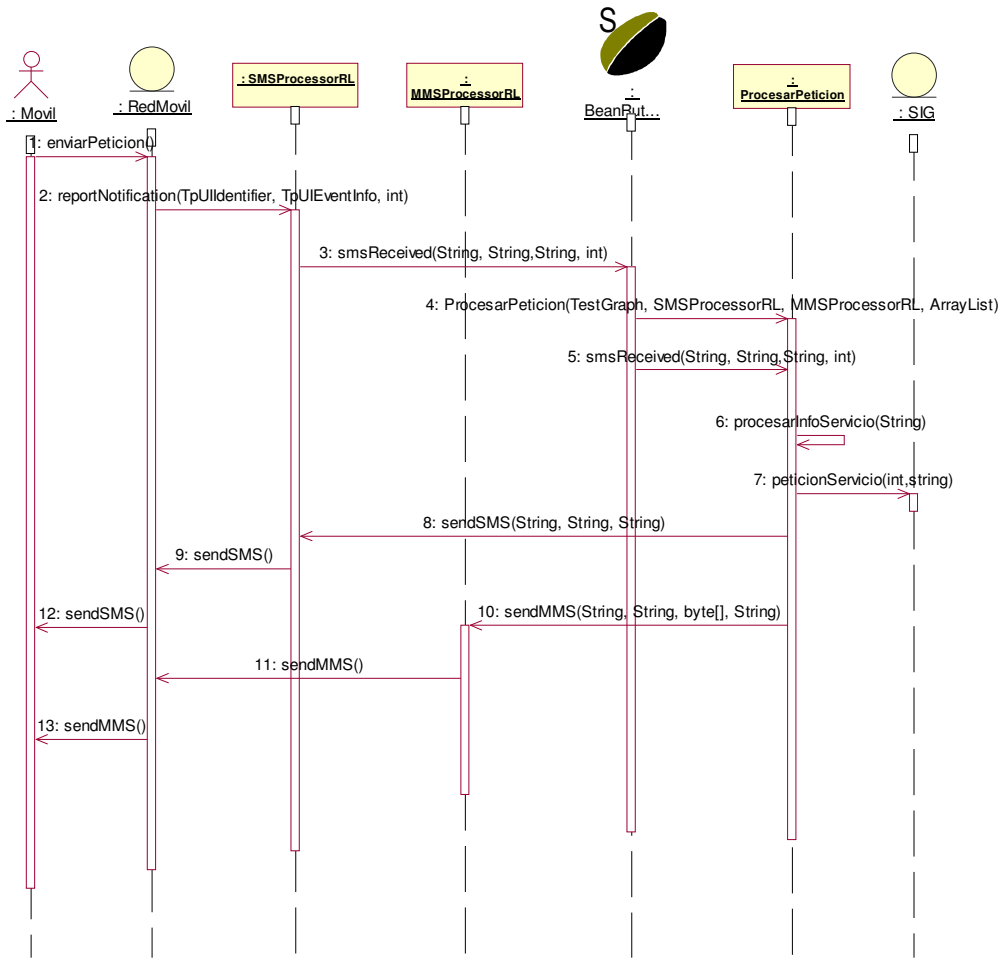

Figura 4. Diagrama de Secuencia Enviar Petición

Este diagrama muestra la secuencia de enviar un mensaje con la solicitud de un servicio y como es procesado por el sistema para generar una respuesta a la petición.

Se representa la red móvil como una entidad que recibe la petición directamente del dispositivo móvil, esta solicitud es reconocida por el API Parlay/OSA como un reporte de notificación de la llegada de un mensaje en la clase SMSProcessor, que llama al método smsReceived del EJB del servicio pasándole los datos correspondientes al número del origen (quien envía el mensaje), el destino (en este caso es el código asignado al servicio el que recibe el mensaje), el contenido del mensaje (ubicación del

usuario y la información necesaria para el servicio) y un identificador aleatorio generado en la clase SMSProcessor para distinguir cada una de las peticiones dentro del sistema desarrollado.

En el EJB se crea un objeto de ProcesarPetición por cada una de las solicitudes con el fin de poder manejarlas simultáneamente, para la creación de este objeto se le pasan como parámetros al constructor de la clase un objeto de la clase TestGraph (que define todos los componentes de la red como puntos, puntos de interés y las líneas o calles de las vías), un objeto de la clase que procesa los mensajes de texto, un objeto que procesa los mensajes multimedia y si el servicio lo requiere se le pasa también un arreglo con los temas que servirán de fondo para generar una imagen para un mensaje multimedia; luego el EJB llama al método smsReceived de ProcesarPetición pasándole también el origen, destino y contenido del mensaje junto con el identificador; esta clase es responsable de analizar el contenido del mensaje y seleccionar de éste la información que debe pasarse al SIG para la solución del servicio, por lo que representa la interfaz de comunicación entre el sistema de mensajería y el SIG. El análisis de la información del mensaje se realiza mediante un llamado al método procesarInfoServicio de esta misma clase, luego, se llama al método peticionServicio que es el encargado directamente de manipular las clases del SIG para generar la respuesta necesaria y devolver una cadena con la que se creará un mensaje. Dentro del método smsReceived y después de haber procesado los datos en el SIG, se llama al método sendSMS de la clase SMSProcessor pasándole como parámetros tres cadenas, una con el número de origen (que ahora sería el correspondiente al servicio), otra con el número del destino (correspondiente al número del dispositivo que realizó la solicitud) y otra con el contenido del mensaje de respuesta; de aquí el mensaje llega a la red móvil que se encarga de enrrutarlo al dispositivo que envió inicialmente el mensaje para el servicio. Si se requiere, se puede generar una imagen o un archivo multimedia con el que se puede enviar un MMS.

- **Caso de uso No. 3**: *Detener Servicio*

  Iniciador: Administrador

  Precondición: El servicio debe existir y debe haber sido activado

  Propósito: Permitir detener el servicio con el fin de hacer mantenimiento o mejoras.

  Resumen: Este caso de uso empieza cuando el administrador mediante una interfaz desactiva el servicio en el servidor de aplicaciones.

  Flujo principal:

  1. El administrador o prestador de servicio presiona el botón que desactiva el servicio.
  2. El sistema destruye todos los objetos, se detiene la notificación del mensaje SMS, el procesador, el gestor de servicios, se inhabilita el framework y se para el rastreo de parlay.

  Flujos alternos:

  - Ninguno

- **Diagrama de Secuencia**

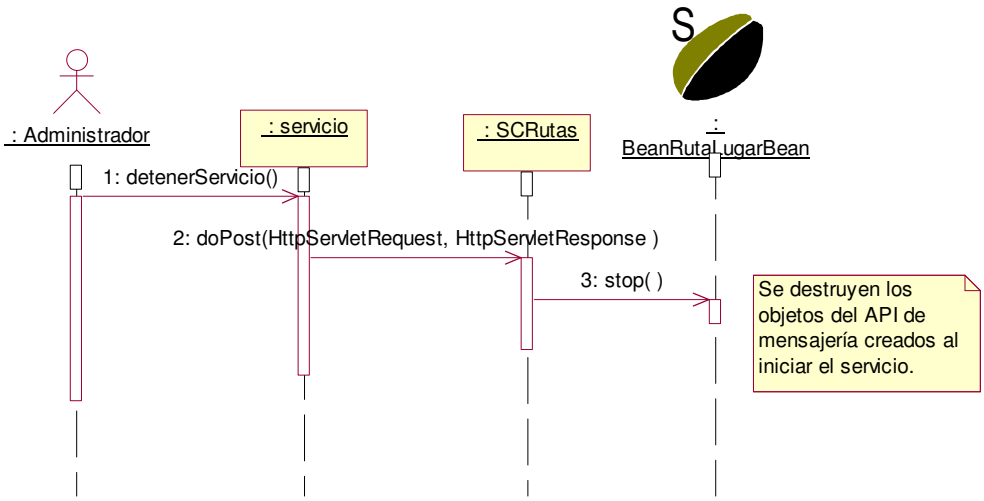

Figura 5. Diagrama de Secuencia Detener Servicio

Para que el administrador pueda detener el servicio se necesita que haya sido activado previamente, al seleccionar el botón de detener en la interfaz gráfica, se llama al método doPost de la clase SCRutas donde se analiza que la opción seleccionada es detener la prestación del servicio, esta clase llama al método stop del EJB que desecha los objetos de procesamiento de mensajes y termina todos los procesos de comunicación con la red móvil.

- **Diagrama de Clases**

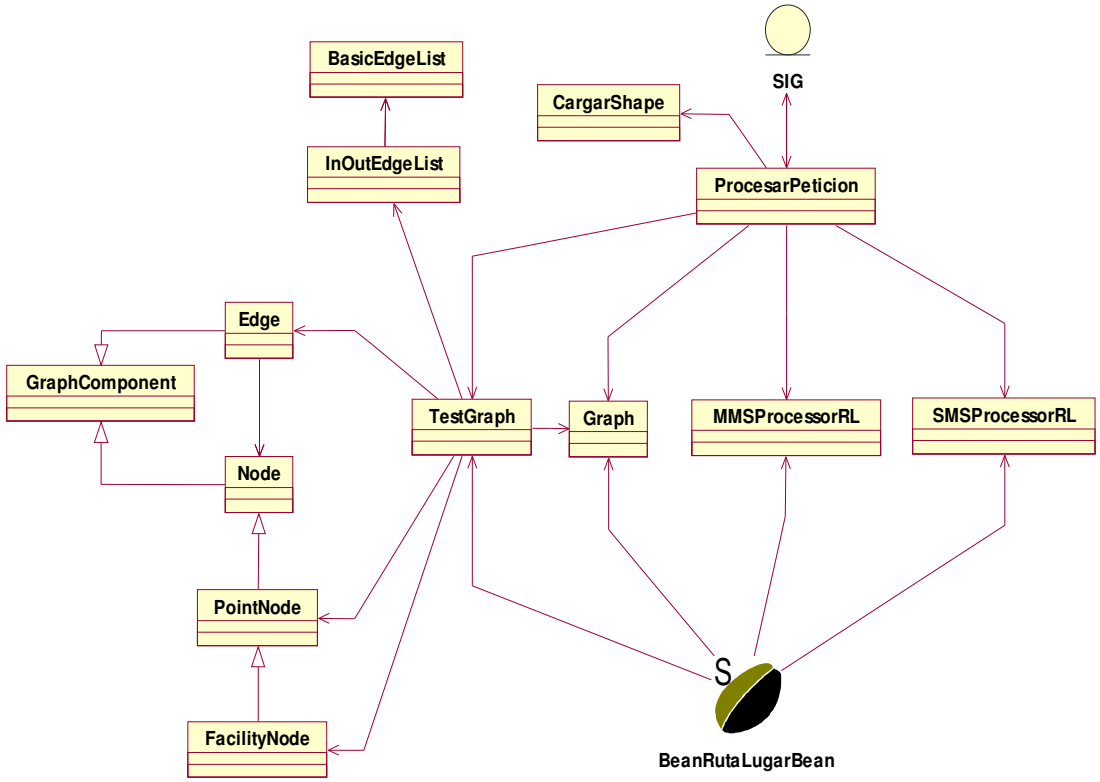

Figura 6. Diagrama de Clases de la Plataforma de Desarrollo

En este diagrama se muestran las clases que intervienen en el funcionamiento de la aplicación, por tratarse de la plataforma como tal, es preciso representar al SIG como una entidad que procesa los datos necesarios, produciendo una respuesta dependiendo del tipo de servicio. No es conveniente explotar las relaciones del SIG, debido a que es éste módulo el que representará la diferencia entre lo que es la plataforma por si sola y el sistema en general (entendamos el sistema como lo correspondiente a la Plataforma + Servicios). En el módulo de servicios se ilustrará como es el efecto del SIG, ya no visto como una entidad sino como un conjunto de clases que interactúan entre si.

- **Diagrama de Despliegue**

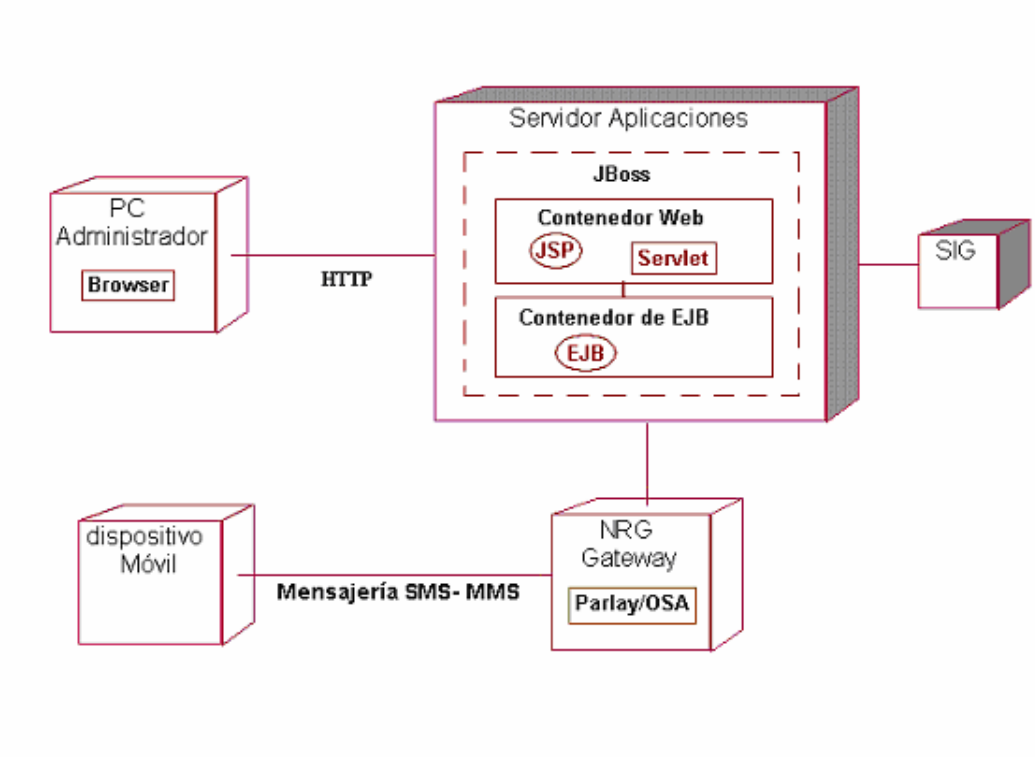

Figura 7. Diagrama de Despliegue Plataforma de Desarrollo

Este diagrama muestra la disposición física de los nodos de procesamiento que componen el sistema. En el caso de la plataforma se tiene el dispositivo móvil que accede a los servicios de localización residentes en el servidor de aplicaciones. La comunicación entre estos nodos se realiza a través del API de mensajería de Parlay/OSA. El nodo del administrador se comunica con el servidor de aplicaciones mediante el protocolo http.

2. SERVICIOS

En esta segunda parte se ilustra el complemento indispensable de la plataforma, debido a que son los servicios en sí los que permiten demostrar en definitiva el correcto desempeño de la plataforma. Al igual que con la plataforma de desarrollo la ilustración de los casos de uso, los de secuencia y el de clases se hacen indispensables para su entendimiento.

Para poder continuar, es conveniente aclarar lo siguiente:

- Los diagramas y la descripción de los casos de uso de la PLDM y de los Servicios son iguales.

- Cada servicio dependiendo de su implementación tiene una interacción interna diferente dentro del Sistema de información geográfico, este comportamiento se ve reflejado en el respectivo diagrama de secuencia.

- Dentro de los servicios el SIG es comprendido como un conjunto de clases y no como una entidad, como era en el caso de la Plataforma.

- La plataforma tiene implementados tres servicios denominados así: Ruta Lugar, Ruta Lugar Cerca y Hallar Dir.

2.1 Ruta Lugar (encontrar la ruta hacia un lugar específico)

Para utilizar este servicio, el usuario debe enviar un mensaje con su ubicación correspondiente al número de calle y carrera donde se encuentra y el lugar a donde quiere llegar; el destino puede ser un lugar específico identificado por el nombre, o una dirección compuesta también por calle y carrera. Este mensaje debe enviarse al código 211 que fue el escogido para este servicio. Ejemplo, si se quiere conocer la ruta desde la Calle 3 con Carrera 7 hasta la Catedral, el mensaje quedaría así: ***c6k7,catedral,.***

El sistema recibe la petición y genera un mensaje de texto con las indicaciones de las calles por las que debe desplazarse el usuario para alcanzar el destino, y un mensaje multimedia que contiene una imagen con la ruta dibujada junto con los nombres de las calles.

A continuación puede verse la figuras representativas a la petición del servicio *Ruta lugar* (código 211) con su respectiva respuesta.

Figura 8. Petición servicio Ruta Lugar (Origen Calle 3 Cra. 7, Destino La Catedral)

Figura 9. Respuesta SMS y MMS a la petición de Ruta Lugar

- **Diagrama de Secuencia**

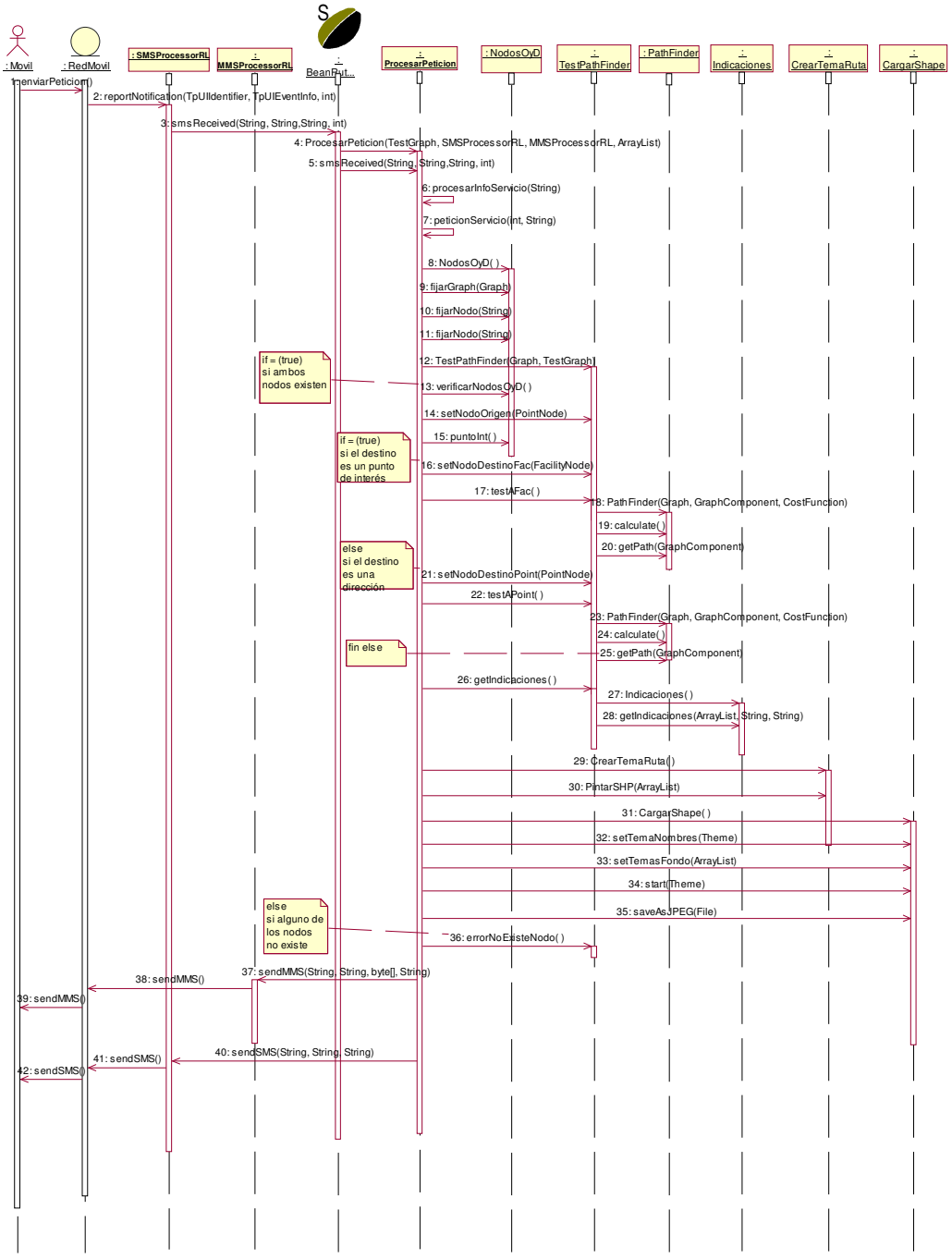

Figura 10. Diagrama de Secuencia enviar petición (*servicio Ruta Lugar*)

El procesamiento de la información para el envío de la solicitud del usuario es igual al explicado en la descripción de la plataforma, solo se va a definir la parte que no se repite, esta es la correspondiente a la secuencia seguida a partir del SIG.

Después que la clase ProcesarPetición ha analizado la información que debe ser enviada al SIG para la prestación del servicio, se llama al método peticionServicio pasándole como parámetros un identificador para diferenciar los datos correspondientes a esta solicitud y la ruta donde debe ser almacenada la imagen que se genere con la ruta hacia el lugar escogido; el identificador de la petición se usa también para armar el nombre de esta imagen, de esta manera cada imagen generada tiene un nombre diferente.

Para este servicio se crea un objeto de la clase NodosOyD al que se le asocia la red de la zona de prueba, posteriormente se fijan las cadenas de los nombres de los nodos de origen y destino en esta clase. Después se crea un objeto de la clase TestPathFinder que es el que accede directamente a la clase del API que permite encontrar la ruta entre dos puntos, teniendo este objeto, ahora es necesario verificar en el objeto de la clase NodosOyD si los nodos fijados existen en la red, para esto se llama al método verificarNodos que devuelve un booleano, verdadero si ambos nodos existen o falso si alguno no se encontró en la red, ya sea que el usuario envió mal la información o que el nodo no esté dentro del rango de la zona de prueba.

Con el fin de analizar de forma detallada, se supone inicialmente que los dos nodos se encuentran en la red, por lo que se prosigue a fijar los nodos en la clase TestPathFinder, primero se fija el nodo origen mediante el método setNodoOrigen que pasa un objeto del nodo como parámetro; para fijar el nodo de destino, es necesario verificar si el usuario desea ir a una dirección o a un punto de interés, por lo que se analiza la variable puntoInt (verdadero si el destino es un punto de interés) de la clase donde se verificó la existencia de los nodos, de ser así se llama al método setNodoDestinoFac (Fac de facility que equivale en español a punto de interés) de la clase TestPathFinder que recibe como parámetro el objeto de la clase FacilityNode correspondiente al destino escogido por el usuario, con esto ya se tienen los nodos de origen y destino y se procede a hallar la ruta más óptima entre estos dos puntos llamando al método testAFac donde se crea el objeto de la clase PathFinder del API de geotools2 pasándole como parámetros la red donde

debe buscar la mejor ruta, el nodo de origen y una función que debe usar para hallar el costo de cada ruta, en este caso se analiza la longitud o distancia que tiene cada calle por lo que la ruta más óptima en este servicio es la más corta, se hace esta aclaración porque se le pueden añadir otros atributos de costo como el tráfico que tiene cada vía o el número de carriles, por ejemplo. Después de crear el objeto de esta clase, se llama al método calculate verifica si el origen es un nodo o un edge (segmento de vía) , hecho esto se llama al método getPath pasándole el nodo de destino, con esto se obtiene la lista de nodos por la que pasa la ruta seleccionada como la más corta. Este proceso se realiza de la misma manera si el destino es una dirección y no un punto de interés. Ya teniendo los nodos por los que se debe seguir, internamente en el método testAFac (para punto de interés) o testAPoint (para una dirección) se obtienen los segmentos de las vías correspondientes a cada par de nodos por los que pasa la ruta, estos segmentos se almacenan en un arreglo para pasarlo al llamar al método getIndicaciones de la clase TestPathFinder, donde se crea un objeto de la clase indicaciones (creada en su totalidad por los desarrolladores de la plataforma) y luego llama al método getIndicaciones de la clase Indicaciones pasándole el arreglo de líneas de la ruta con las que se deben generar las indicaciones, el nombre del punto de inicio y el nombre del destino. Hasta este punto ya se tienen una cadena con las indicaciones que es devuelta por el método testAFac o testAPoint de la clase TestPathFinder a la clase ProcesarPeticion.

En el caso que alguno de los nodos no se encontrara dentro de la red se llama al método errorNoExisteNodo de la clase TestPathFinder que lo que hace es devolver una cadena informando que alguno de los nodos no es correcto o no se encuentra dentro de la zona de prueba.

Ya sea con la cadena de indicaciones o con la cadena de error, es posible en este momento generar un mensaje de texto que puede ser enviado al usuario como respuesta.

Para este servicio, se desarrollo la funcionalidad de generar una imagen con el mapa de la ruta, por lo que después de generadas las indicaciones y como ya se tenía el nombre del archivo donde se almacenará la imagen, se crea un objeto de la clase CrearTemaRuta que toma los puntos por donde esta pasa y dibuja las líneas que unen estos nodos y que finalmente representan las calles, esto se realiza al llamar al método pintarSHP donde se

pasa un arreglo con las coordenadas de los puntos correspondientes a las esquinas o cruces de las vías de la ruta. Posteriormente, se crea un objeto de la clase cargarShape que carga el tema con los nombres de las calles mediante el método setTemaNombres, los temas de las vías y de los puntos de interés mediante el llamado al método setTemasFondos y finalmente en el método start que recibe como parámetro el tema de la ruta va a cargar todos estos mapas en una ventana de visualización, después se llama al método saveAsJPEG pasándole como atributo la ruta del archivo donde se debe guardar la imagen y posteriormente se puede generar el mensaje multimedia que contiene esta imagen para mandársela al usuario.

- **Diagrama de Clases**

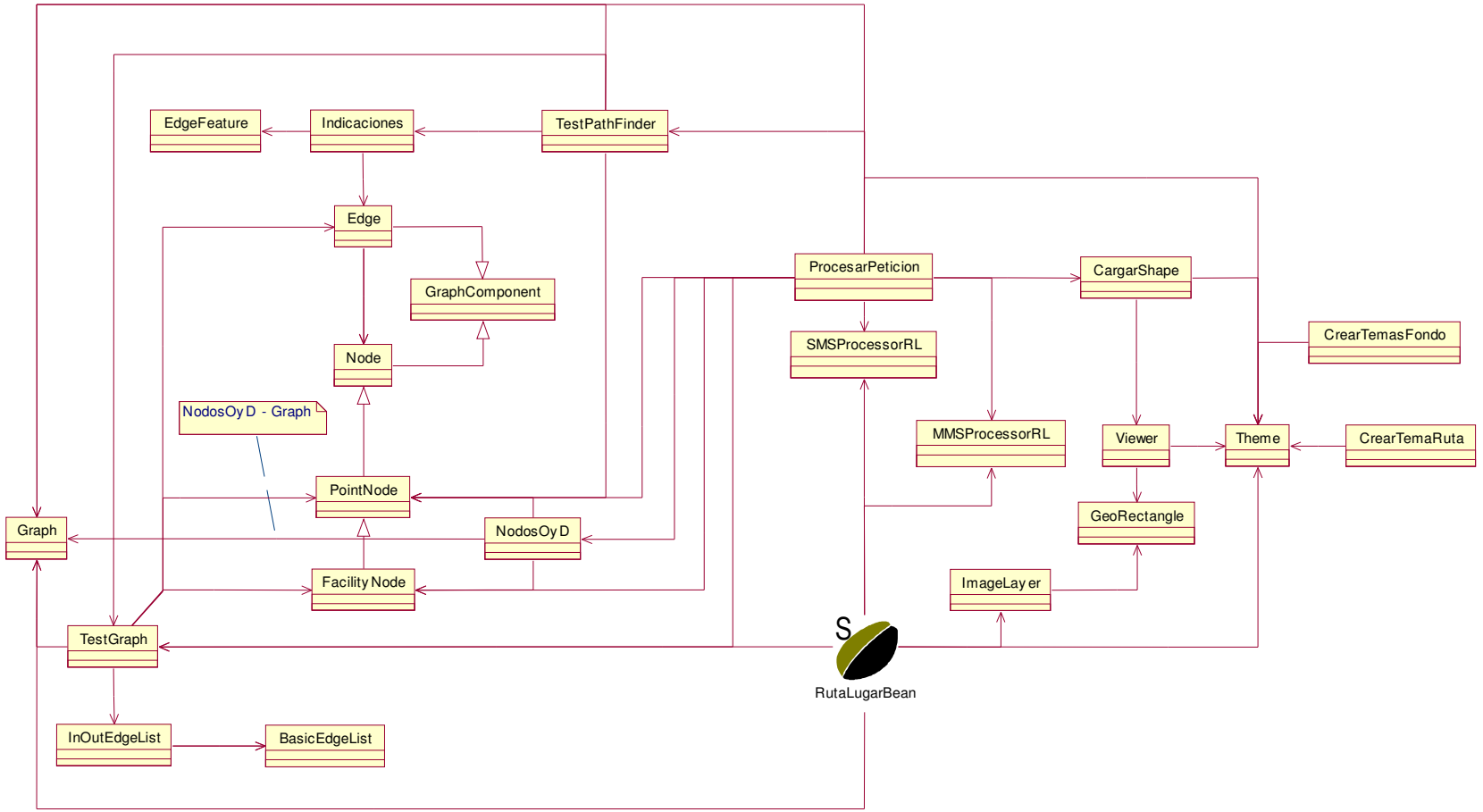

Figura 11. Diagrama de clases - servicio Ruta Lugar

En este diagrama (figura 11) se ilustran las clases que intervienen en la realización del servicio de prueba (servicio ruta lugar), también se encuentran las clases de la aplicación como tal, debido a que se debe mostrar la funcionalidad de todo el sistema, todas las clases ya fueron descritas en el capítulo anterior. Se debe tener en cuenta que en el diagrama de clases de la aplicación se representaba el SIG como una entidad, ya que las clases que se usaran dependen del servicio que se este describiendo, por lo que en la figura 11 se representan las clases correspondientes al SIG que influyen en el funcionamiento del servicio de hallar la ruta hacia un lugar.

2.2 Ruta Lugar Cerca (Ruta hacia el lugar más cercano)

Para utilizar este servicio, se necesita la ubicación del usuario de igual forma que para el servicio de ruta hacia un lugar; la lógica cambia para el manejo del destino, que para este caso el usuario debe enviar dentro del mensaje de texto de la solicitud el tipo de lugar al que se quiere dirigir y no la dirección o el nombre; como tipo de lugar se puede tomar por ejemplo museo, iglesia, supermercado, etc. El mensaje con los datos de origen y destino son enviados al código 212 asignado para este servicio. Como respuesta a esta petición, el sistema genera un mensaje de texto con las indicaciones hacia el lugar más cercano y uno multimedia que contiene la imagen de la ruta.

Por ejemplo, un usuario se encuentra en la calle 4 con carrera 5 y desea ir al museo más cercano, entonces el mensaje dirigido al código 212 debe estar escrito de la siguiente manera: **c4k5,museo,.** A continuación se ilustra la petición del servicio Ruta Lugar Cerca con su respectiva respuesta, en este caso particular el museo mas cercano es el *Guillermo León Valencia* .

Se podrá visualizar la similitud de los mensajes recibidos (SMS y MMS) entre el servicio Ruta Lugar y Ruta Lugar Cerca, debido a que la forma de presentarle al usuario la respuesta de ambos servicios es la misma, lo que se diferencia es el procesamiento interno que involucra a cada servicio.

Figura 12. Petición del servicio Ruta Lugar Cerca

Figura 13. Respuesta SMS y MMS al servicio Ruta Lugar Cerca (Código 212)

- **Diagrama de Secuencia**

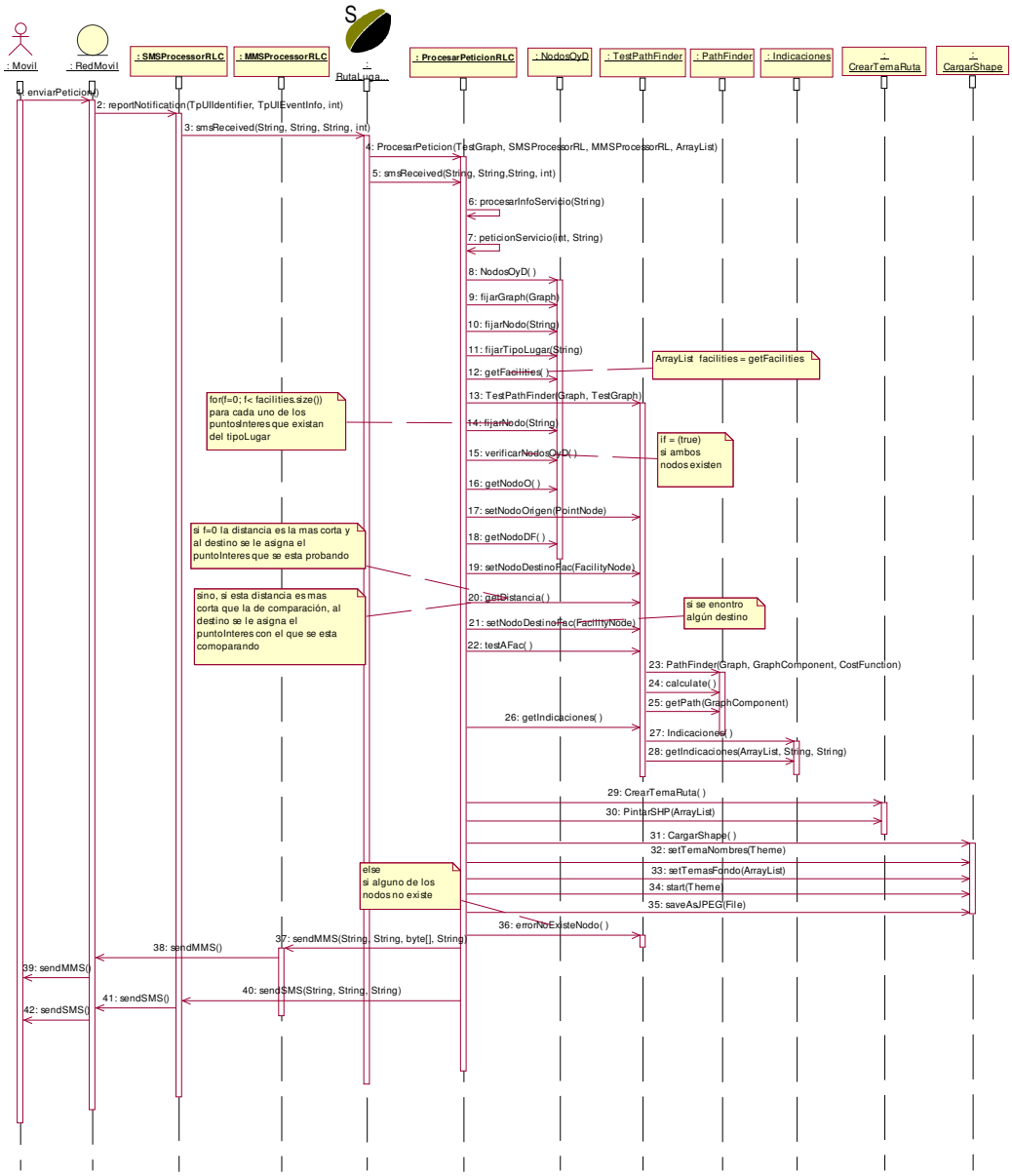

Figura 14. Diagrama de secuencia enviar petición (Ruta Lugar Cerca)

Este diagrama muestra la secuencia de enviar un mensaje con la solicitud del servicio que permite hallar la ruta hacia el lugar más cercano de acuerdo al tipo de lugar escogido por el usuario y como es procesado por el sistema para generar una respuesta a la petición.

Se representa la red móvil como una entidad que recibe la petición directamente del dispositivo móvil, esta solicitud es reconocida por el API Parlay/OSA como un reporte de notificación de la llegada de un mensaje en la clase SMSProcessorRLC, que llama al método smsReceived del EJB del servicio pasándole los datos correspondientes al número del origen (quien envía el mensaje), el destino (número que debe recibir el mensaje, que en este caso es el asignado al servicio), el contenido del mensaje (ubicación del usuario y la información necesaria para el servicio) y un identificador aleatorio generado en la clase SMSProcessorRLC para distinguir cada una de las peticiones dentro del sistema desarrollado. En el EJB se crea un objeto de ProcesarPeticiónRLC por cada una de las solicitudes con el fin de poder manejarlas simultáneamente, para la creación de este objeto se le pasan como parámetros al constructor de la clase un objeto de la clase TestGraph (que define todos los componentes de la red como puntos, puntos de interés y las líneas o calles de las vías), un objeto de la clase que procesa los mensajes de texto, un objeto que procesa los mensajes multimedia y debido a que al final se genera una imagen con la ruta, se pasa también un arreglo con los temas que servirán de fondo para generar dicha imagen; luego, el EJB llama al método smsReceived de la clase ProcesarPeticion pasando los siguientes parámetros (String remitente, String destinatario, String contenidoMensaje, int identificadorPeticion). En éste método, utilizando el identificador de la petición, se crea el nombre del archivo donde se almacenará la imagen generada más adelante; después, se llama al método procesarInfoServicio pasándole como parámetro el contenido remitido en el mensaje de texto para que se separe cada uno de los datos que lo componen y que están separados por comas. Seguidamente se almacena en la variable indicaciones la cadena que se devuelve al llamar el método peticionServicio que recibe los parámetros: identificador, nombreImagen (nombre del archivo donde se guardará la imagen.

El método peticionServicio es de suma importancia, ya que es donde se realiza el procesamiento del servicio haciendo uso del SIG, se define de la siguiente manera:

public String peticionServicio(int identificador, String nombreImg). Se definen las siguientes veriables:

double distancia = 0. Almacena la menor distancia encontrada entre las que se van hallando.
double distComparar. Guarda cada distancia que se va encontrando para compararla con la menor distancia encontrada, si es menor, a la variable distancia se le asigna el valor de *distComparar*.
String i = "". Almacena la cadena que se le devuelve al método que llamó inicialmente a este método y que luego es enviada al usuario como respuesta en un SMS. Si se pudo encontrar una ruta, esta variable contiene las indicaciones de cómo llegar al destino, en el caso contrario, contiene una oración con el fin de informar que alguno de los datos enviados es incorrecto.

Después de la definición de las variables necesarias, se realiza el procesamiento en el SIG, creando primero un objeto de la clase NodosOyD donde se verifica que los nodos y/o lugares solicitados por el usuario se encuentren dentro de la zona definida en la red de vías definida en la plataforma. Los métodos de esta clase que permiten procesar esa información son:

nodosOyD.fijarGraph(graph). Fija el objeto de la red donde se deben buscar los lugares o puntos y donde luego se busca la mejor ruta.
nodosOyD.fijarNodo(origen). Como se tiene conocimiento del origen de la ruta, se fija el punto de origen que es donde está ubicado el usuario.
nodosOyD.fijarTipoLugar(tipoLugar). Como en este servicio el usuario informa a que tipo de lugar le interesa dirigirse y no el lugar específico al que quiere ir, se fija la variable tipoLugar, que permite encontrar dentro de la red los lugares que califican como de interés para el usuario y se almacenan en una lista o arreglo.
ArrayList facilities =nodosOyD.getFacilities(). Con la llamada a este método se obtienen todos lo puntos que van a ser analizados para encontrar el más cercano. La variable se define como facilities ya que su traducción hace referencia a puntos de interés.

Una vez que se tienen los datos correspondientes al origen y los posibles destinos, se debe encontrar cual de ellos es el más cercano, por lo que se necesita analizar cada una de las rutas para ver cual implica recorrer la menor distancia; esto se realiza en la clase TestPathFinder que es la que permite encontrar las rutas de una forma básica (encuentra las esquinas por donde debe pasar el usuario) y las distancias, por esto se crea un objeto de esta clase pasándole al constructor los datos correspondientes a la red donde se debe buscar la ruta y un objeto de la clase TestGraph que contiene cada uno de los elementos con los que se construyó la red como son las vías y los puntos correspondientes a las esquinas y a los puntos de interés.

Ahora se debe encontrar la ruta más corta, por lo que se halla la distancia para cada uno de los lugares almacenados en la lista *facilities* obtenida anteriormente. Para realizar esto, se implementa un ciclo for desde cero hasta el número de elementos contenidos en la lista menos uno (de esta forma se analizan todos los elementos porque se inicia desde cero). Dentro de este ciclo se realizan las siguientes operaciones:

nodosOyD.fijarNodo(facilities.get(f)). Fija el elemento f en la clase NodosOyD.
Se realiza un ciclo que verifica si el origen fijado anteriormente y el destino fijado en la línea anterior se encuentran dentro de la zona definida en la plataforma, el método que hace este proceso es el siguiente:
if (nodosOyD.verificarNodosOyD()) Verifica que el origen y el destino existan en la red.

Si lo anterior se cumple entonces se obtiene los objetos como nodos del origen y destino, ya que para verificarlos se usaban las cadenas correspondientes a sus nombres enviadas en el mensaje por el usuario.

- *nodoOrig = nodosOyD.getNodoO()*. Se almacena el objeto del nodo correspondiente al punto de origen.
- *testPathFinder.setNodoOrigen(nodoOrig)*. Se fija el nodo en la clase que permite hallar la ruta
- *nodoDestPrueba = nodosOyD.getNodoDF()*.Se almacena el objeto del nodo correspondiente al punto de destino (DF en el método porque el Destino es FacilityNode lo que significa que no es una esquina sino un punto de interés)

- *tpf.setNodoDestinoFac(nodoDestPrueba)*. Se fija el nodo destino correspondiente a un punto de interés en la clase que permite hallar la ruta.
- *distComparar = tpf.getDistancia()*. Este método permite calcular la distancia que se debe recorrer desde el origen hasta el destino que se esta analizando. Este valor se guarda en una variable para ser comparado con el mínimo que se ha encontrado.
- *if (f == 0)*. Si es la primera distancia que se encuentra (lo que significa que se esta analizando el destino almacenado en el primer lugar de la lista)

    - *distancia = distComparar*. La distancia encontrada equivale a la menor hallada.
    - *destino = (String) facilities.get(f)*. Como es el destino hasta donde se tiene que recorrer la menor distancia hasta ahora, se obtiene el nombre del lugar que hasta ahora es el más cercano.
    - *nodoDest = nodoDestPrueba*. Al objeto del nodo destino se le asigna el nodo que se esta brobando, ya que como se explico corresponde al mas cercano hasta el momento.

- *else if (distComparar < distancia)*. Si no es el primer destino analizado, se debe verificar si la distancia hasta encontrada para este elemento es menor que la que se tiene almacenada como la más corta, de ser así:

    - distancia = distComparar. A la distancia que se tiene como menor se le asigna la que se esta analizando.
    - *destino = (String) facilities.get(f)*. A la cadena que contiene el nombre del destino al que se debera dirigir el usuario se le asigna el nombre del lugar que se esta analizando y cuya distancia es la menor hasta el momento.
    - *nodoDest = nodoDestPrueba*. Se almacena el nodo correspondiente a este lugar en el objeto correspondiente al nodo de destino.

En el caso que la distancia que se está probando no es menor que la que se tiene como referencia, no se hace nada, ya que la distancia menor seguirá siendo la que se tenía.

Después de verificar que el punto de origen se encuentra en la red de vías y que se analizaron los posibles destinos, se debe examinar que se haya encontrado un destino, ya que puede ser que el usuario haya escrito mal el tipo de lugar al que se quiere dirigir o que no esté considerado dentro de las opciones del servicio. Para ver cuales son los tipos de lugares tenidos en cuenta en el desarrollo de la plataforma, dirigirse al anexo C correspondiente al manual de usuario, que contiene una lista de los lugares con el respectivo tipo de lugar.

if (!destino.equals("")). Si la cadena que almacena el nombre del destino más cercano no está vacía, significa que el tipo de lugar escogido si estaba dentro de las opciones del servicio y existían lugares de ese tipo en la red de vías de la zona del centro de la ciudad, por lo que se tiene entonces hasta el momento el origen y el destino de la ruta que debe seguir el usuario hasta el lugar mas cerca.

- *tpf.setNodoDestinoFac(nodoDest)*. Se fija el nodo de destino en la clase que permite hallar la ruta, anteriormente ya se había fijado el origen.
- *tpf.testAFac()*. Método que permite encontrar la ruta y las calles por las que esta pasa.
- *i = testPathFinder.getIndicaciones()*. Método que permite obtener las indicaciones y las devuelve para ser almacenadas en la cadena i que se usará como contenido del mensaje de texto de respuesta a la petición del servicio. En este método de la clase TestPathFinder se crea un objeto de la clase Indicaciones y luego se llama al método getIndicaciones pasándole como parámetros la lista de calles por las que pasa la ruta y el nombre del origen para que se generen las indicaciones, luego estas se devuelven en una cadena que se asigna a la variable i.

- *CrearTemaRuta tr = new CrearTemaRuta()*. Se crea un objeto de la clase que pinta el tema de la ruta.
- *Theme tRuta = tr.PintarSHP(tpf.getDatos())*. Se llama al método que pinta cada línea correspondiente a las calles que componen la ruta pasándole las coordenadas de las esquinas.

- *cshp = new CargarShape()*. Objeto que permite visualizar todos los temas generados para crear la imagen que formará el mensaje MMS
- *cshp.setTemaNombres(tNombres)*. Se añade a la vista el tema de los nombres de las calles para que el usuario pueda saber a que calle o carrera corresponde cada línea de la imagen.
- *cshp.setTemasFondo(temas)*. Se añaden los temas de fondo como son el que contiene todas las calles del centro de la ciudad y la que contiene los puntos de interés.
- cshp.init(). Método que inicializa algunas variables.
- cshp.start(tRuta). Método que permite cargar en la vista los temas que fueron añadidos anteriormente, porque no se habían desplegado o mostrado, solo se habían añadido a la clase.
- cshp.saveAsJPEG(new File(nombreImg)). Con este método se guarda la imagen visualizada en una pequeña ventana como una imagen .JPEG.

En el caso que no se hubiera encontrado alguno de los puntos, se debe crear un mensaje que informe al usuario que alguno de los datos que hacían parte del mensaje de texto de la solicitud era incorrecto. Para esto se llama al método de la clase TestPathFinder que devuelve una cadena con un mensaje de error.

errorNoExisteNodo(). Este mensaje se guarda en la variable i que se devuelve al método smsReceived que fue el que inicialmente lo llamó para que formar con esta cadena el mensaje de texto que se envía al usuario.

Luego se devuelve la variable del mensaje, ya sea que contenga el mensaje con las indicaciones o el mensaje de error. Con el contenido de esta variable se llama al método sendSMS de la clase SMSProcessor para que se envíe el mensaje al usuario.

Si todos los datos enviados por el usuario eran correctos, se pudo generar una ruta y se puede enviar un mensaje con la imagen, por lo que se crea un mensaje multimedia con la imagen y se llama al méto sendMMS de la clase MMSProcessor para que se envíe al usuario.

- **Diagrama de Clases**

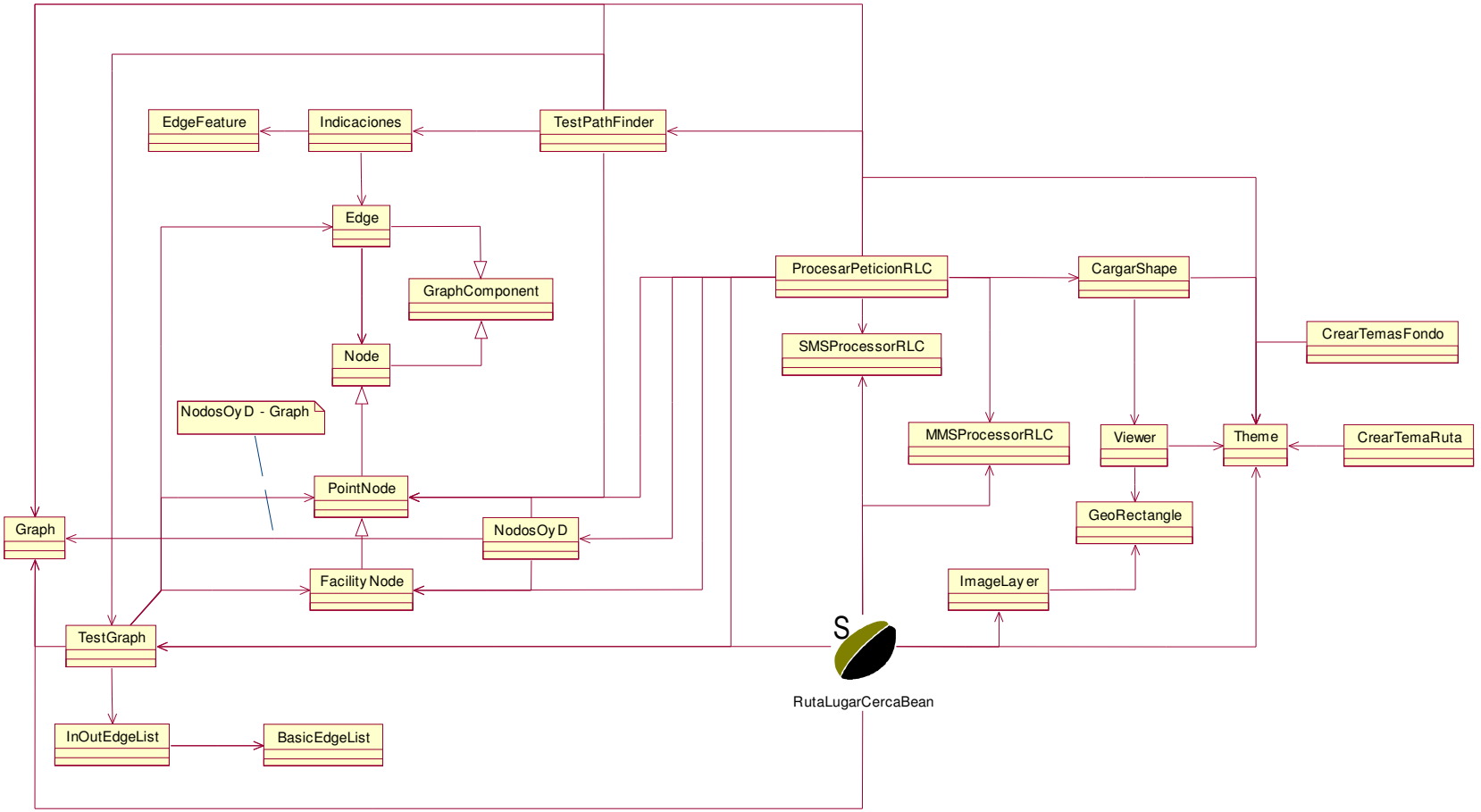

Figura 15. Diagrama de Clases - enviar petición (Ruta Lugar Cerca)

Cada servicio tiene clases propias que ayudan a procesar la información. Algunas de las clases tienen prácticamente el mismo contenido para cada servicio, como por ejemplo las clases que procesan los mensajes de texto y multimedia llamadas inicialmente SMSProcessor y MMSProcessor, pero para cada servicio se les añade al final las iniciales del servicio del que reciben la petición, por lo que para este servicio se llamarán SMSProcessorRLC y MMProcessorRLC las últimas letras corresponden a Ruta hacia el Lugar más Cercano, la única diferencia entre las clases de este tipo para cada servicio es que tienen referencia a un objeto del EJB del servicio.

Existen otras clases que deben ser implementadas por cada servicio como la que procesa la petición, ésta también tiene al final las iniciales del servicio del que procesa la información para este servicio se llama ProcesarPeticionRLC. Debe existir una clase de este tipo diferente para cada servicio, ya que el modo de procesar la información varía según los parámetros recibidos del usuario en el contenido del mensaje, así, en este servicio se reciben la ubicación actual y el tipo del lugar a donde se quiere dirigir el usuario, mientras que para el servicio ruta hacia un lugar, los datos son la ubicación actual y la dirección o nombre del lugar a donde se quiere dirigir.

2.3 Hallar Dir (Hallar dirección):

Para utilizar este servicio, el usuario debe enviar un mensaje que contenga el nombre del lugar del cual desea averiguar la dirección; aunque este servicio no requiere la posición del usuario se hace uso de la red de vías generada en la plataforma para buscar la dirección. El sistema devuelve como respuesta un mensaje de texto con la dirección completa del sitio escogido. El código del mensaje es 213.

A diferencia de los dos servicios anteriores, éste, tiene como único parámetro el lugar destino del cual queremos conocer su dirección, además que como respuesta sólo se recibe un mensaje SMS, el mensaje multimedia no se aplica para este servicio. Un ejemplo de petición del servicio *Hallar Dir* para conocer la dirección del Supermercado Ley sería: **ley,.** A continuación se ilustra el mensaje de petición del servicio (Código 213) y su respetiva respuesta.

Figura 16. Petición del servicio Hallar Dir

Figura 17. Respuesta SMS al servicio Hallar Dir (Código 213)

- **Diagrama de Secuencia**

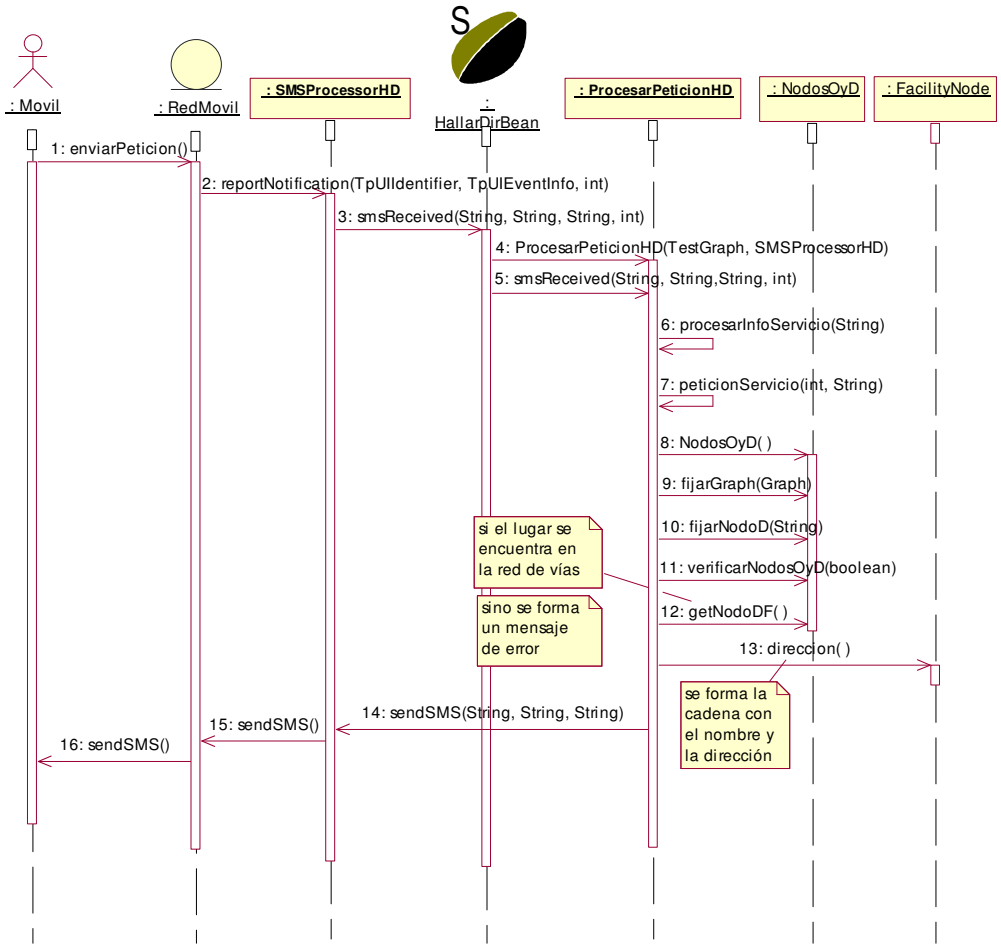

Figura 18. Diagrama de secuencia enviar petición (Hallar Dir)

La secuencia ilustrada en el diagrama anterior es igual a la de los diagramas anteriores hasta que el sistema interactúa con la clase que procesa la petición en este caso se llama ProcesarPeticionHD debido a que se le añaden las iniciales del servicio del cual procesa las solicitudes. Cuando el sistema recibe un mensaje y llama a la clase ProcesarPetición, le pasa como parámetros solo el objeto de la clase que crea los elementos de la red de vías y el objeto de la clase que procesa los mensajes de texto; luego se llama al método smsReceived de esta clase pasándole los siguientes datos: remitente, destinatario (código asignado al servicio 213), el contenido del mensaje y el identificador de la petición. Dentro de este método se realiza el siguiente procesamiento:

procesarInfoServicio(aMessageContent). Método que analiza los datos enviados en el contenido del mensaje de texto de la solicitud.
direccion = peticionServicio(id). En la cadena dirección se almacena el nombre y la dirección del lugar solicitado por el usuario, esta cadena es devuelta en el llamado al método petición servicio que contiene sigue la siguiente secuencia:

String i=""; cadena donde se almacenará la dirección del lugar solicitada en el servicio.
nodosOyD = new NodosOyD(); objeto de la clase que permite verificar si los lugares o puntos solicitados por el usuario se encuentran dentro de red de vías.
nodosOyD.fijarGraph(graph); se pasa el objeto de la red donde se busca el lugar y donde se tienen almacenados los atributos de cada elemento de la red de vías como los lugares y las calles.
nodosOyD.fijarNodoD(destino); se fija un solo lugar, ya que no se tiene un origen y un destino sino solo el lugar del cual se va a obtener la dirección.
if (nodosOyD.verificarNodosOyD(true)). Si el nodo o lugar existe se obtiene un objeto correspondiente a ese elemento del cual se pueden obtener todas sus características como nombre, tipo de lugar, dirección, etc.

- *FacilityNode lugar = nodosOyD.getNodoDF().* Se obtiene el objeto correspondiente al lugar.
- i= destino+": "+lugar.direccion(). Se obtiene el atributo correspondiente a la dirección y se forma la cadena i que contiene como destino el nombre del lugar enviado por el usuario y con el método lugar.direccion se obtiene su dirección.
- return i; se devuelve esta cadena que va a ser usada como contenido del mensaje de texto de respuesta para el usuario.

En el caso que no se encuentre el lugar ya sea porque esté fuera del área de prueba tomada en la plataforma (que es el centro de la ciudad de Popayán) o porque el usuario escribió mal el nombre del lugar en el mensaje de texto, se genera un mensaje de error.
else

- i = "El lugar del cual se necesita la direccion no existe o se escribio mal";
- return i. se devuelve la cadena con el mensaje de error para ser usada como contenido del mensaje de texto de respuesta.

Después de haber procesado la información en las clases correspondientes al

SIG, se llama al método:

unSMSProcessor.sendSMS("213", aSender, direccion). Método encargado de enviar los mensajes de texto, recibe como parámetros:

213: Remitente (código del servicio)
aSender: Destinatario (número del móvil que envió la solicitud)
direccion: contenido del mensaje de texto (nombre y dirección del lugar o mensaje de error)

- **Diagrama de Clases**

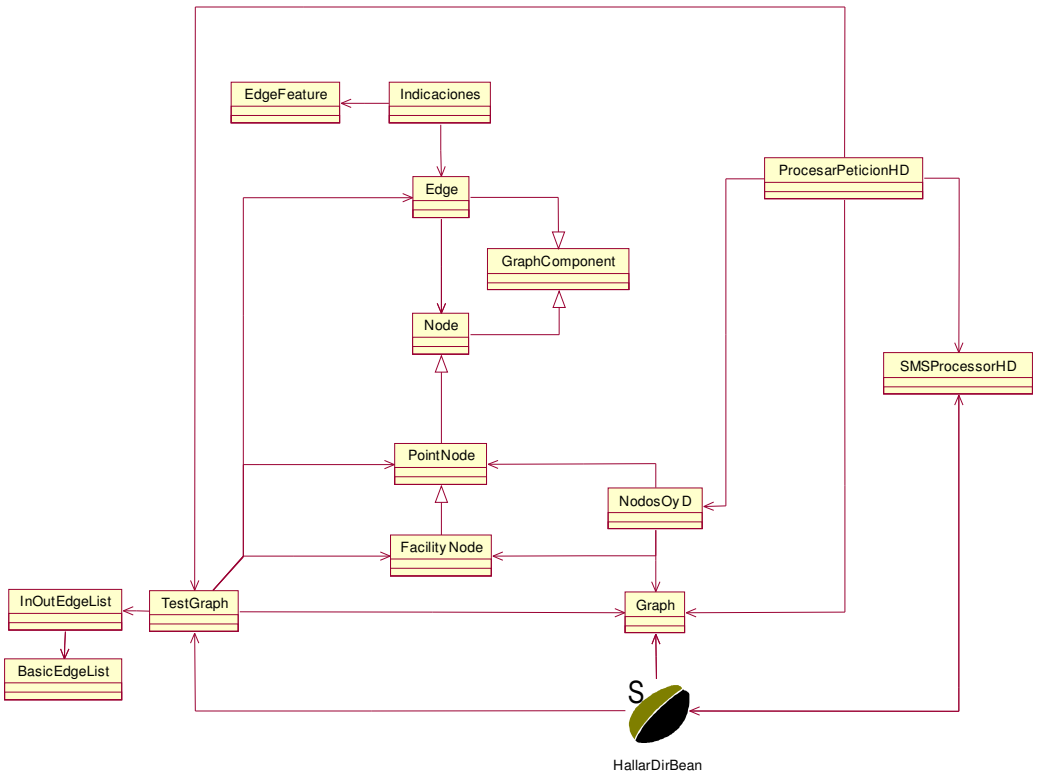

Figura 19. Diagrama de Clases - enviar petición (Hallar Dir)

Para este servicio se intervienen menos clases que para los otros, ya que no se genera un mensaje MMS porque lo que se pide es una dirección, por esto, no se necesita manejar temas ni capas de imagen, solo se tienen las clases directamente relacionadas con la creación de la red y del envío de mensajes de texto.