Diagramas

En este documento se describe textualmente y mediante diagramas ,el comportamiento de las clases que componen el proyecto.

Se han utilizado para el desarrollo de los diagramas los siguientes símbolos:

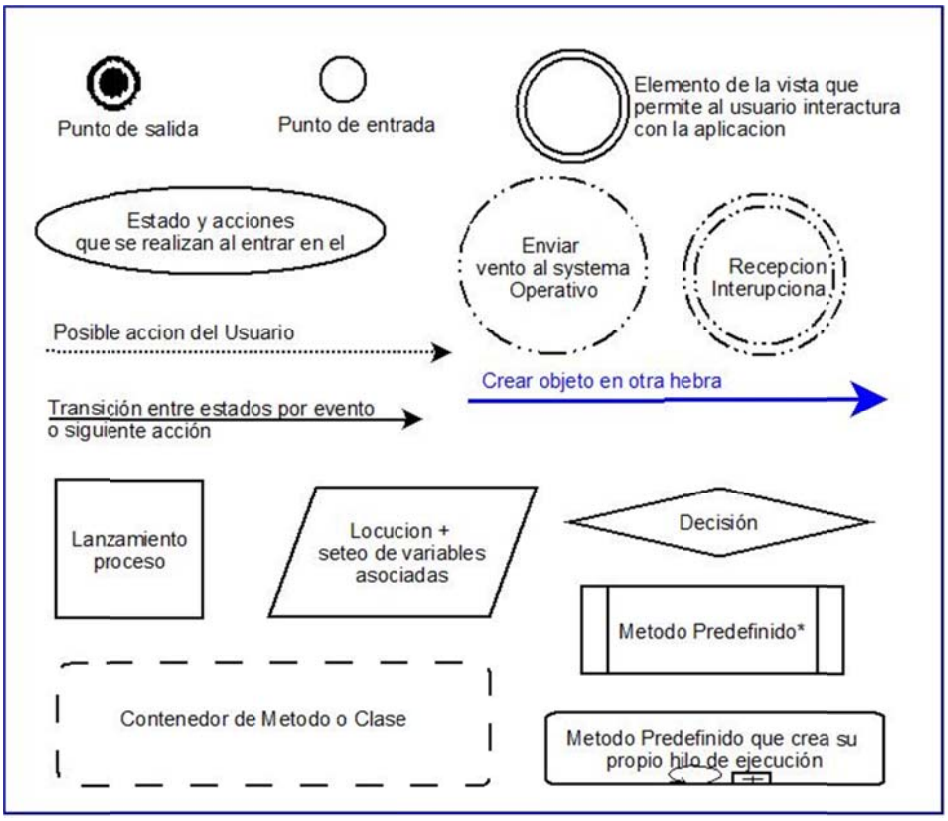

Ilustración 1: Leyenda

Dado que no se trata de un código secuencial, sino que el comportamiento de la aplicación, depende de los eventos: del sistema operativo, del comportamiento del usuario y de la respuesta dada por los sensores. En los diagramas que se muestran a continuación, se representa este hecho, mediante puntos de entrada de eventos y líneas discontinuas, que solo se siguen en caso de que se produzca el evento.

Dado la complejidad del comportamiento de cada una de las clases, no se detallan completamente en un solo diagrama, sino que los métodos más largos aparecen nombrados dentro de un recuadro, el cual se describe a posteriori. Tampoco se describen todos los procesos, los procesos estándar, propios de la clase de la que se hereda se describen con palabras, para no extender en exceso este apartado.

La aplicación arranca ejecutando la clase “UserActivity.java”, que comienza comprobando si es la primera vez que se ejecuta la aplicación, si es así crea la base de datos. Luego comprueba el último modo de trabajo guardado, si es modo instalador lanza “InstalerActivity.java” finalizando la ejecución de “UserActivity.java”.
Si estamos en modo Usuario, se comprueba si el servicio está o no arrancado, para mostrar el botón de arrancar o de parar, que al pulsarse crean el servicio o le envían un mensaje para pararlo. También se muestra un botón para cambiar a modo Instalador, que lanza un popup en el que se solicita la introducción de una clave, si la clave se

introduce correctamente, se detiene el servicio, en caso de estar encendido y se lanza “InstalerActivity.java”.

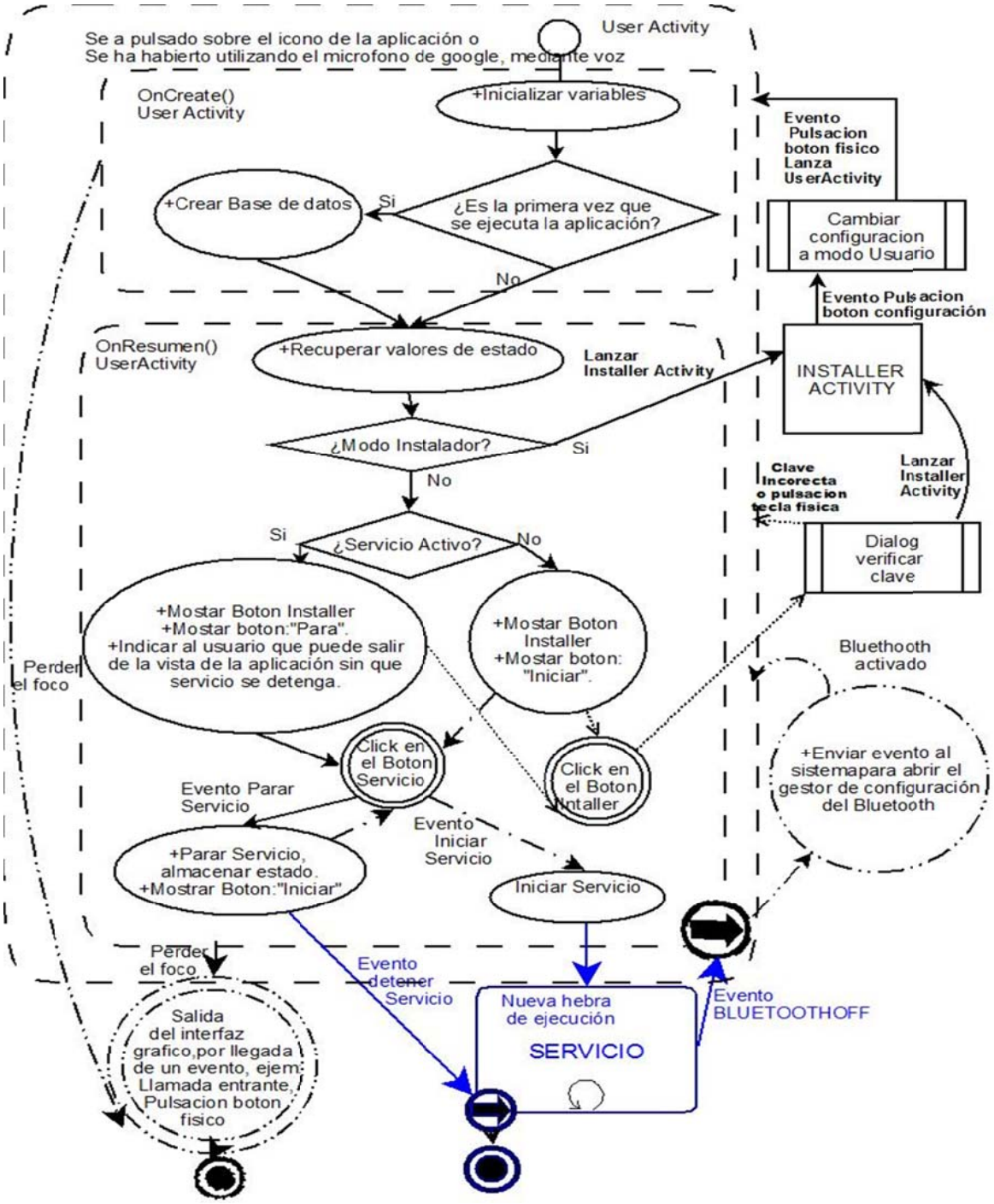

Ilustración 2: User Activity

En el siguiente diagrama se muestra agrandes rasgos la lógica del Servicio de detección de balizas y de locución de los mensajes, que se ejecuta al crear un objeto de la clase “SpeechBluService.java”. Esta clase se crea una segunda hebra de ejecución con un ciclo de vida independiente de “UserActivity.java”, y será destruida únicamente al apagarse el teléfono o recibir un evento de parada originado por una instancia de “UserActivity.java”. Esta hebra a su vez, crea otros hilos de ejecución. Crea un hilo de

ejecución corto de duración 0,8 segundos cada segundo, durante el cual el sensor de Bluetooth detecta balizas. Lanza otro hilo de ejecución cada vez que ha de reproducir una cadena de texto y el método “OnDeviceDetected”, al detectar una baliza por primera que no se encuentra registrada en la base de datos, llama a “HTTP_JSON_POST.java” que crea otro hilo de ejecución para comunicarse con el servidor remoto. Los métodos recuadrados se describen a lo largo de este capítulo.

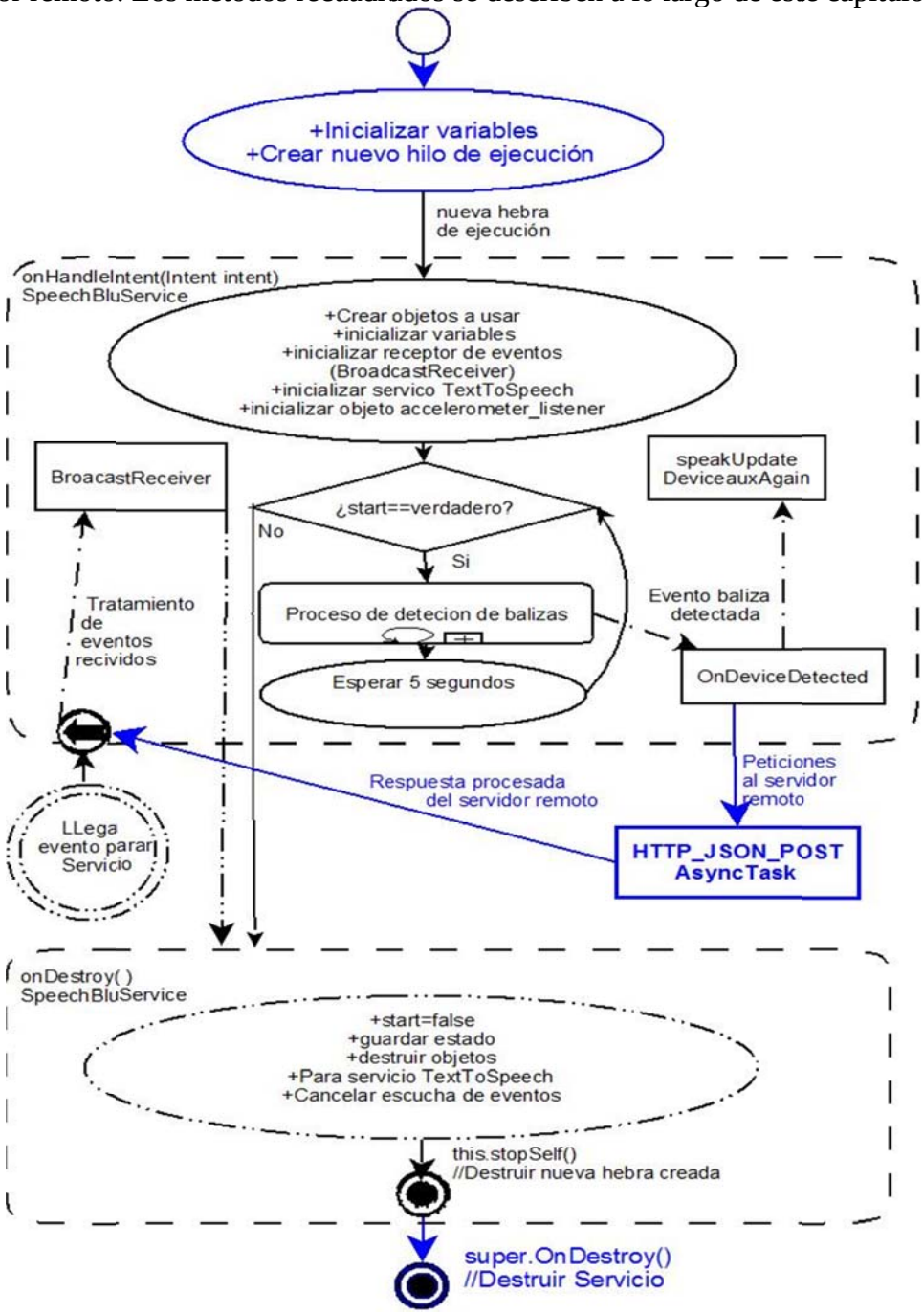

Ilustración 3: Visión global de la clase SpeechBluService

El método “OnDeviceDetected” cuando se detecta una baliza (el proceso de detección de balizas es una sucesión de llamadas a métodos de la clase “BluetoothManager” que ha sido extraído de **¡Error! No se encuentra el origen de la referencia.**]). En primer lugar se comprueba si la baliza está en la lista negra (lista compuesta por balizas que no están registradas en el sistema), si lo esta se ignora y si no se comprueba si ha sido detectada anteriormente durante la ejecución actual del servicio, si

es así se lanza el método “speakUpdateDeviceauxAgain”. En otro caso, se consulta en la base de datos si hay información reciente de la baliza. Si no hay información o no lo suficientemente reciente, se crea el objeto “HTTP_JSON_POST”, para solicitarta al servidor remoto. En otro caso,, se añade a la lista de balizas actual y se comprueba si el tiempo que se tarda en reproducir el mensaje es superior a 5 segundos, si es así se ejecuta “speakUpdateDeviceauxAgain”. Y en caso contrario, se comprueba si el nivel de señal recibido es superior al mínimo que tiene asociado la baliza, solo entonces, se reproduce el texto asociado a la baliza.

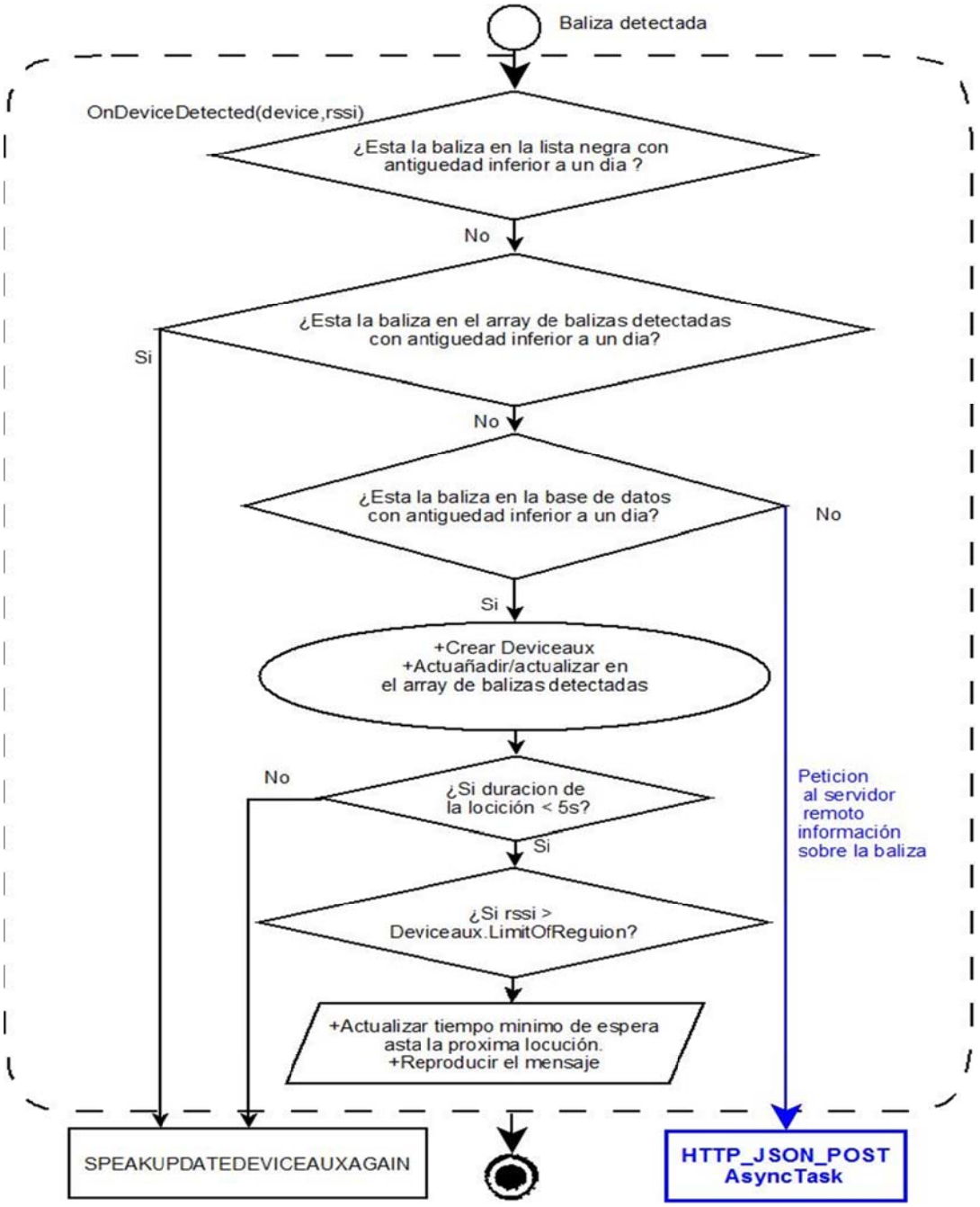

Ilustración 4: OnDeviceDetected

HTTP_JSON_POST.java , es la clase encargada de la comunicación con el servidor remoto. Se crean objetos de esta clase cuando no se dispone de información reciente sobre una baliza, o se quiere actualizar/registrar los datos asociados a una de ellas. El

cuero de las peticiones http es construido utilizando la clase “OrionJsonManager.java”, también se utiliza esta clase, para crear objetos: Device, Deviceaux o DeviceDAO a partir de las respuestas del servidor remoto que serán enviados a.

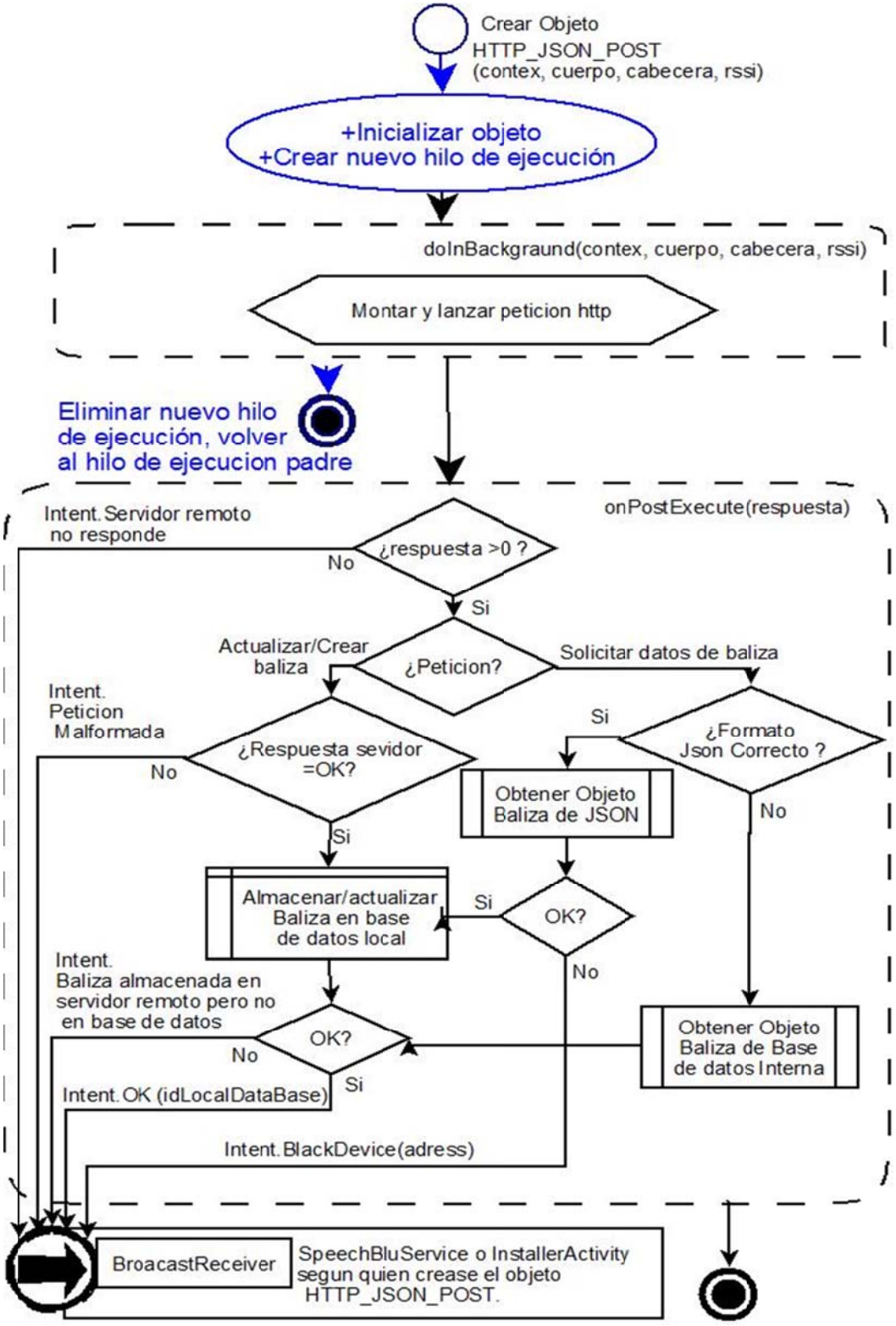

Ilustración 5: HTTP_JSON_POST

El método “speakUpdateDeviceAuxAgain”, junto con la clase “DeviceAux.java” es donde más tiempo se ha invertido durante el desarrollo de este proyecto, pues conjuntamente se encarga de reproducir las locuciones cuando una baliza ya ha sido detectada anteriormente, indicando la dirección del movimiento. El método sigue la siguiente lógica:

Si aún no se ha anunciado anteriormente la baliza, se comprueba si el nivel de señal detectada es superior al mínimo asociado a la baliza y si el mensaje asociado a la baliza, tiene una duración inferior a 5 segundos, en tal caso se reproduce el mensaje, marcando la baliza como anunciada.

Si la baliza ya ha sido anunciada y a trascurrido el tiempo mínimo (duración último mensaje reproducido) tras la última locución de la app y tras la última locución de la baliza (tiempo mínimo=duración mensaje baliza), entonces:

- Si la media de los niveles de señal recibidos de la baliza es superior a 56dBm y no se ha indicado recientemente que se esté muy cerca de ella (*count< 1),* entonces comprobamos la duración del mensaje:
    - Si es superior a 5 segundos, se establece el tiempo mínimo a esperar antes de la próxima petición de locución igual a la duración del mensaje más tres segundos; se envía el mensaje a la cola de reproducción y se marca *count=0* .
    - Si es inferior a 5 segundos, se sobrescribe el mensaje a reproducir con “estas muy cerca de”, más el mensaje asociado a la baliza; se marca como tiempo mínimo a esperar antes de la próxima petición de locución, la duración del nuevo mensaje; se envía el mensaje a la cola de reproducción y se marca *count=2*.
    - Si nos estamos acercando, se sobrescribe el mensaje a reproducir con “acercándote a”, más el mensaje asociado a la baliza; se marca como tiempo mínimo a esperar antes de la próxima petición de locución, la duración del nuevo mensaje; se envía el mensaje a la cola de reproducción y se marca *count=count -1*.
    - Si nos estamos alejando. Se comprueba si la media de las señales recibidas por la baliza es inferior al minimo.
        - Si es inferior, se sobrescribe el mensaje a reproducir con “salistes de”, más el mensaje asociado a la baliza; se marca como tiempo mínimo a esperar antes de la próxima petición de locución, la duración del nuevo mensaje; se envía el mensaje a la cola de reproducción y se marca *count=2*.
        - Si es superior, se sobrescribe el mensaje a reproducir con “alejándote de”, más el mensaje asociado a la baliza; se marca como tiempo mínimo a esperar antes de la próxima petición de locución, la duración del nuevo mensaje; se envía el mensaje a la cola de reproducción y se marca *count=count -1*.

El esperar a que termine la locución anterior, antes de encolar un nuevo mensaje en el reproductor, se realiza para evitar la reproducción tardía; pues es confuso si te indican

que te estas alejando, cuando en realidad te estas acercando pues acabas de cambiar la dirección de tu movimiento recientemente.

La variable *count*, tiene doble funcionalidad. Por un lado, cuando estas realizando pequeños movimientos muy cerca de la baliza, impide que el mensaje “estas muy cerda de”, se reproduzca continuamente. Al igual que en el caso de estar realizando pequeños movimientos a la distancia en que se percibe un nivel de señal aproximado al nivel se señal mínimo asociado a la baliza, en este caso impide que se esté reproduciendo continuamente el mensaje “saliste de”.

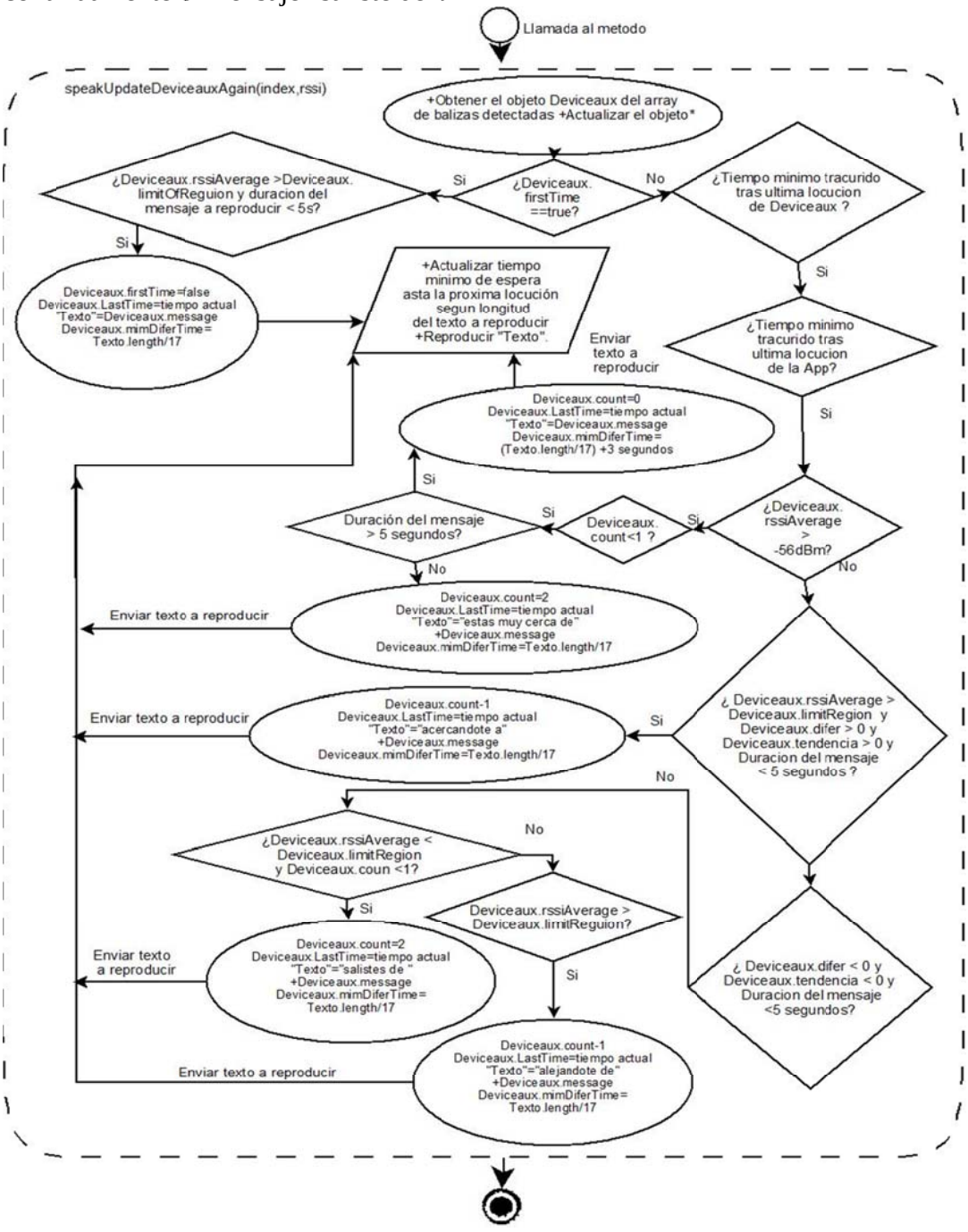

Ilustración 6: speakUpdateDeviceAgain

Por ultimo muestro el funcionamiento en Modo instalador. “InstalerActivity.java”, lo primero que hace al ejecutarse, es recuperar las variables de estado y comprobar que el

ultimo estado guardado, fue modo Instalador. Si no es así, cierra la actividad y lanza "UserActivity.java". Si estamos en modo instalador, muestra la vista correspondiente en la cual:

- Se puede acceder al menú de configuración donde se permite cambiar: la URL del servidor de soporte y el modo de funcionamiento.
- Se puede escribir el mensaje a asociar a una baliza.
- Buscar balizas cercanas y seleccionar una, para asociarle el mensaje escrito y el nivel de señal que se esté recibiendo, como nivel de señal mínimo aceptable para la reproducción del mensaje.
- Y observar la respuesta del servidor.

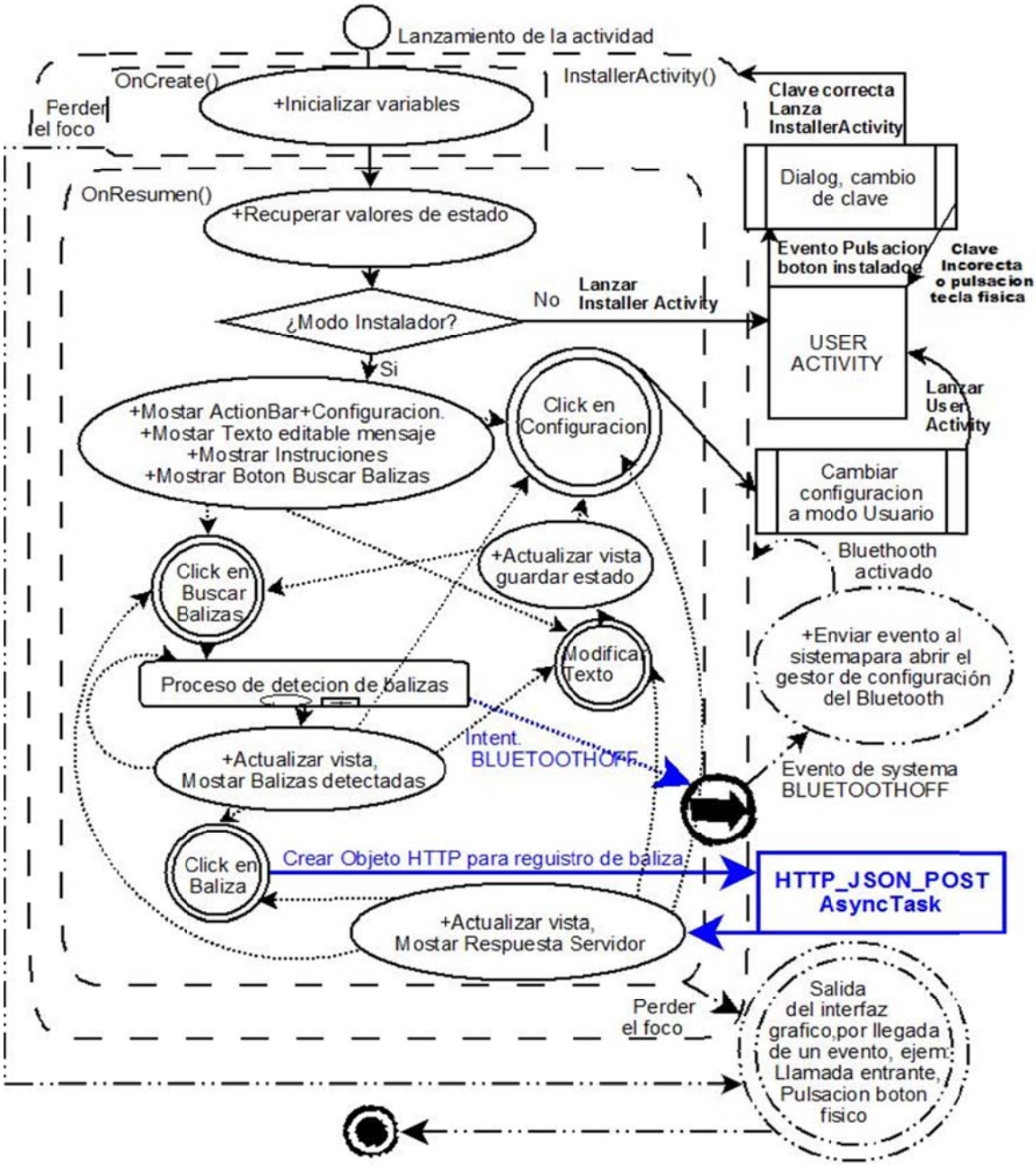

Ilustración 7: InstallerActivity