

1 Diagrammi di sequenza

In questa sezione vengono descritte e rappresentate tramite diagrammi di sequenza UML le sequenze di azioni ritenute più significative con lo scopo di facilitare la comprensione delle comunicazioni tra oggetti facenti parte dell'applicativo Android_g. Per quest'ultimo motivo i diagrammi di sequenza non rappresentano l'effettiva realtà ma una versione semplificata e che non rifletterà in tutto l'implementazione.

1.1 Avvio Service per il rilevamento beacon

Il diagramma in figura ?? rappresenta l'avvio del service_g che si occupa del rilevamento dei beacon_g funzionalità focale dell'intero applicativo.

La classe `NavigationManagerPresenter` invoca il metodo `startService()` su `NavigationManagerImp`, all'interno del metodo viene istanziato un oggetto `intent` di tipo `Intent` necessario per creare effettivamente un bind service_g, `BeaconManagerAdapter`, attraverso la chiamata del metodo `bindService()`, passando come parametro `intent`. Nella fase di creazione del service_g di tipo `BeaconManagerAdapter` viene chiamato il metodo `onCreate()` nel quale viene creata un'istanza della classe `BeaconManager` offerta dalla libreria `AltBeacong`. Si effettuano inoltre diverse chiamate per il settaggio e la configurazione di `beaconManager` che non sono rappresentate per mantenere il diagramma più leggibile. Una volta settato `beaconManager` l'oggetto `beaconManagerAdapter` si mette in ascolto di `beaconManager` chiamando il metodo `setMonitorNotifier` iniziando la fase di monitoring_g.

A questo punto `beaconManagerAdapter` è un listener di `beaconManager` il quale una volta rilevata la region dei beacon in cui il device si trova scatena l'evento `didEnterRegion()` notificando i propri listener, ossia l'oggetto di tipo `beaconManagerAdapter`.

Individuata la region tramite l'evento `beaconManagerAdapter` effettua un controllo per capire se la region è riconosciuta dall'applicativo, se lo è `beaconManagerAdapter` entra nella fase di ranging_g, in cui saranno raccolti dettagliatamente i dati di tutti i beacon rilevati. `beaconManagerAdapter` si mette in ascolto in modalità ranging di `beaconManager` tramite la chiamata del metodo `setRangeNotifier()`.

A questo punto `beaconManagerAdapter` riceve l'evento di rilevazione beacon attraverso il metodo `didRangeBeaconsInRegion()` il quale restituisce una `Collection` di `Beacon` e la `Region` di appartenenza.

Per la gestione degli elementi all'interno della `Collection` si rimanda al diagramma successivo.

1.2 Elaborazione beacon rilevati e comunicazione broadcast

Il diagramma in figura ?? rappresenta l'interazione che avviene tra i componenti dell'applicativo allo scopo di rilevare dettagliatamente i dati trasmessi dai beacon circostanti al device.

L'oggetto di tipo `BeaconManagerAdapter` è un `service`, e implementa il listener di `BeaconManager`: `RangeNotifier` il quale scatenerà, dopo una scansione, l'evento `didRangeBeaconsInRegion()` passando come parametri una `Collection` di Beacon rilevati e la `Region` di appartenenza. I parametri vengono elaborati da `BeaconManagerAdapter` il quale dopo aver creato una `PriorityQueue` costruisce un wrapper, (`MyBeacon`) di ogni Beacon aggiungendolo alla `PriorityQueue` tramite `add()`.

Una volta elaborati tutti i Beacon ricevuti `BeaconManagerAdapter` crea un messaggio `Intent` in cui inserisce la `PriorityQueue` tramite la chiamata del metodo `putExtra()`. Costruisce l'oggetto `LocalBroadcastManager` per utilizzarlo nella chiamata del metodo `sendMessageBroadcast()` che si occuperà di inviare l'`Intent` in altre parti dell'applicazione costruite appositamente per ricevere il messaggio ed elaborarlo, queste parti estenderanno la classe `BroadcastReceiver` offerta dal SDK Android.

1.3 Avvio navigazione

Il diagramma in figura ?? rappresenta il flusso d'eventi generato nelle classi del model qualora si richiedesse l'avvio della navigazione. La richiesta parte da `NavigationManagerPresenter` con la chiamata del metodo `startNavigation()` sull'oggetto `NavigationManagerImp` passando come parametri la destinazione identificata dall'oggetto di tipo `PointOfInterest`. Il `NavigationManagerImp` si occupa quindi di impostare il grafo all'oggetto di tipo `NavigatorImp` con il metodo `setGraph()` dopodiché invoca il metodo `calculatePath()` in cui è calcolato il percorso da seguire durante la navigazione attraverso l'oggetto `DijkstraPathFinder` che restituisce una `List` di `EnrichedEdge` salvata in `navigator` in un campo dati. A questo punto `navigator` è pronto per restituire le informazioni (`ProcessedInformation`) richieste dalla classe `NavigationManagerImp`, quest'ultimo invoca il metodo `toNextRegion()` passando come parametri la lista di beacon_g rilevati e ricevuti tramite l'oggetto `BroadcastReceiver`. `navigator` ricava dai beacon_g rilevati il beacon_g il cui segnale risulta essere il più potente (`getMostPowerfulBEacon()`), quindi controlla che il beacon ritenuto più vicino all'utente appartiene alla region of interest (ROI_g) del percorso previsto, infine costruisce le `ProcessedInformation` richieste grazie all'oggetto `Edge` identificato come prossimo tratto di percorso da percorrere. Le `ProcessedInformation` vengono quindi ritornate a `NavigationManagerImp` che le restituisce a `NavigationManagerImp` il quale le scomatterà e le restituirà alla view e quindi all'utente.

1.4 Avvio della bussola

Il diagramma in figura ?? rappresenta il flusso generato dall'oggetto della classe `NavigationManagerPresenter`, esso effettua due operazioni principali:

- Creazione di `NavigationManager` che causa la creazione dell'oggetto `Compass` a cui viene passato come parametro del costruttore il riferimento a `SensorManager`, classe della libreria Android che permette di recuperare i riferimenti ai sensori del device attraverso la chiamata del metodo `getDefaultSensor(typeSensor)`. `Compass` per calcolare l'orientamento del device necessita dei dati provenienti dal magnetometro e accelerometro.
- `startCompass()` invece accende la bussola `Compass` attraverso la classe `NavigationManager` il quale chiama il metodo `registerListener()`, tale metodo tramite il riferimento a `sensorManager` chiama il metodo `registerListener()` e imposta l'oggetto `compass` observer dei sensori.

1.5 Costruzione di oggetti Table da Json

Il diagramma in figura ?? rappresenta il flusso generato dall'oggetto della classe `RemoteEdgeDao` per la costruzione di un oggetto `EdgeTable` a partire da un oggetto `JsonObject`:

- viene creato un oggetto `Gson`. Tale oggetto permette di gestire l'oggetto `JsonObject` passato;
 - su questo oggetto viene invocato il metodo `fromJson()`, passando come parametri l'oggetto `JsonObject` e l'oggetto `EdgeTable.class`. In questo modo viene creato l'oggetto `EdgeTable`, sfruttando il fatto che l'oggetto `JsonObject` è stato creato utilizzando gli stessi nomi per i campi dato di tale oggetto e i campi dato dell'oggetto `EdgeTable`.
-

Tale costruzione è analoga per oggetti `BuildingTable`, `CategoryTable`, `EdgeTypeTable`, `PhotoTable`, `PointOfInterestTable` e `RegionOfInterestTable`, mentre è differente per la classe `RoiPoiTable`. In questo caso infatti non è stato possibile scaricare un file JSON che abbia dei campi dato con lo stesso nome dei campi dato di `RoiPoiTable`. Il diagramma in figura ?? rappresenta la costruzione di un oggetto `RoiPoiTable` a partire da un oggetto `JsonObject`:

- viene creato un oggetto `GsonBuilder`. Tale oggetto è necessario per la costruzione di un oggetto a partire da un oggetto `JsonObject` che abbia campi dato con nome differente dall'oggetto da costruire;
- viene creato un oggetto di una classe anonima che implementa la classe `JsonDeserializer<RoiPoiTable>`. In tale classe viene fatto l'override del metodo `deserialize()` definendo come un oggetto `RoiPoiTable` può essere costruito sfruttando i campi dati dell'oggetto `JsonObject`;
- viene invocato sull'oggetto `GsonBuilder` il metodo `registerTypeAdapter()` passando come parametri gli oggetti `RoiPoiTable.class` e l'oggetto della classe che implementa `JsonDeserializer<RoiPoiTable>`;
- viene creato un oggetto `Gson` invocando il metodo `create()` sull'oggetto `GsonBuilder`;
- infine, per creare l'oggetto `RoiPoiTable`, viene invocato il metodo `fromJson()` sull'oggetto `Gson` passando come argomenti l'oggetto `JsonObject` e l'oggetto `RoiPoiTable.class`.

1.6 •

Figura 1: Diagramma di sequenza - Avvio di un service_g per il rilevamento beacon

Figura 2: Diagramma di sequenza - Elaborazione beacon rilevati e comunicazione broadcast

Figura 3: Diagramma di sequenza - Avvio navigazione

Figura 4: Diagramma di sequenza - Avvio della bussola

Figura 5: Diagramma di sequenza - Costruzione di EdgeTable da Json

Figura 6: Diagramma di sequenza - Costruzione di RoiPoiTable da Json