CLIPS

Communication & Localization with Indoor Positioning Systems

Università di Padova

Definizione di prodotto v1.00

leaf.gruppo@gmail.com

Versione	1.00
Data Redazione	2016-04-10
Redazione	Marco Zanella
	Oscar Elia Conti
	Andrea Tombolato
	Federico Tavella
	Davide Castello
	Eduard Bicego
Verifica	Cristian Andrighetto
Approvazione	Oscar Conti
Uso	Esterno
Distribuzione	Prof. Vardanega Tullio
	Prof. Cardin Riccardo
	Miriade S.p.A.

Diario delle modifiche

Versione	Data	Autore	Ruolo	Descrizione
1.00	2016-04-10	Oscar Elia Conti	Responsabile di progetto	Approvazione del documento
0.19	2016-04-10	Cristian Andrighetto	Verificatore	Verifica del documento
0.18	2016-04-09	Eduard Bicego	Progettista	Correzioni delle classi ImageDetailActivity, ImageListFragment, ImageListFragmentViewImp, PreferencesViewImp, LoggingView, LoggingViewImp, DeveloperUnlockerViewImp, HelpViewImp, MapDownloaderActivity e LogInformationActivity
0.17	2016-04-09	Eduard Bicego	Progettista	Correzioni minori nei diagrammi di sequenza
0.16	2016-04-08	Federico Tavella	Progettista	Correzioni nel package beacon
0.15	2016-04-08	Cristian Andrighetto	Verificatore	Verifica del documento
0.14	2016-04-08	Marco Zanella	Progettista	Correzioni generali su compenenti del model

Versione	Data	Autore	Ruolo	Descrizione
0.13	2016-04-08	Andrea Tombolato	Progettista	Aggiunte componenti view
0.12	2016-04-08	Oscar Elia Conti	Progettista	Aggiunte componenti presenter
0.11	2016-04-07	Cristian Andrighetto	Verificatore	Verifica del documento
0.10	2016-04-07	Marco Zanella	Progettista	Aggiunte componenti model
0.09	2016-04-06	Eduard Bicego	Progettista	Aggiunta sottosezione Metodo e formalismo di specifica
0.06	2016-04-06	Eduard Bicego	Progettista	Aggiunti diagrammi di sequenza Avvio Service, Ranging Beacons e avvio navigazione
0.05	2016-04-03	Eduard Bicego	Progettista	Completata sezione Standard di progetto
0.04	2016-04-02	Eduard Bicego	Progettista	Aggiornata sezione Introduzione
0.03	2016-03-22	Oscar Elia Conti	Progettista	Aggiunta sezione "Specifica dei componenti"
0.02	2016-03-22	Oscar Elia Conti	Progettista	Aggiunta sezione "Standard di progetto"
0.01	2016-03-18	Oscar Elia Conti	Progettista	Definizione struttura documento

Indice

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Questo documento definisce nel dettaglio la struttura e le relazioni tra le parti del prodotto$_g$, approfondendo ulteriormente dove ritenuto necessario. In particolare vengono descritti in dettaglio i package, le classi e le interfacce, concludendo con il tracciamento tra le classi e i requisiti analizzati nell'*Analisi dei requisiti v4.00* .

1.2 Scopo del prodotto

Lo scopo del prodotto$_g$ è implementare un metodo di navigazione indoor$_g$ che sia funzionale alla tecnologia Bluetooth Low Energy (BLE$_g$). Il prodotto$_g$ comprenderà un prototipo software$_g$ che permetta la navigazione all'interno di un'area predefinita, basandosi sui concetti di Indoor Positioning System (IPS$_g$) e smart place$_g$.

1.3 Glossario

Allo scopo di rendere più semplice e chiara la comprensione dei documenti viene allegato il *Glossario v4.00* nel quale verranno raccolte le spiegazioni di terminologia tecnica o ambigua, abbreviazioni ed acronimi. Per evidenziare un termine presente in tale documento, esso verrà marcato con il pedice $_g$.

1.4 Riferimenti utili

1.4.1 Riferimenti normativi

- capitolato d'appalto C2: CLIPS$_g$: Comunication & Localization with Indoor Positioning Systems: http://www.math.unipd.it/~tullio/IS-1/2015/Progetto/C2.pdf;
- *Norme di progetto v4.00.*

1.4.2 Riferimenti informativi

- Documentazione Android SDK: http://developer.android.com/guide/index.html;
- Documentazione AltBeacon Library: https://altbeacon.github.io/android-beacon-library/documentation.html;

- Documentazione SQLite: https://www.sqlite.org/docs.html;
- Documentazione JavaDoc JGraphT Library: http://jgrapht.org/javadoc/;
- Materiale di riferimento del corso di Ingegneria del Software$_g$ - Diagrammi delle classi: http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E03.pdf;
- Materiale di riferimento del corso di Ingegneria del Softwaere$_g$ - Model View Presenter: http://www.math.unipd.it/~rcardin/sweb/Design%20Pattern%20Architetturali%20-%20Model%20View%20Controller_4x4.pdf;
- Design Pattern: elementi per il riuso di software ad oggetti - Gamma, Helm, Johnson, Vlissides - editore Pearson - 2002;
- UML e ingegneria del software: dalla teoria alla pratica - Luca Vetti Tagliati - 2015.

2 Standard di progetto

2.1 Standard di documentazione del codice

Per gli standard di documentazione del codice si fa riferimento al documento *Norme di progetto v4.00*.

2.2 Standard di denominazione di entità e relazioni

Per tutte le entità e le relazioni valgono gli standard di denominazione seguenti:

- per le entità definite come package$_g$ classi, attributi e metodi è necessario fornire denominazioni chiare e concise;
- per la denominazione delle entità sono da preferire i sostantivi mentre per le relazioni i verbi;
- eventuali abbreviazioni sono preferibilmente da evitare nonostante siano ammesse nei casi in cui siano comprensibili e non ambigue.
- per le regole tipografiche sui nomi si fa riferimento al documento *Norme di progetto v4.00*.

2.3 Standard di programmazione

Per gli standard di programmazione si fa riferimento al documento *Norme di progetto v4.00*.

2.4 Strumenti di lavoro e procedure

Per gli strumenti di lavoro e le procedure per la realizzazione del progetto si fa riferimento al documento *Norme di progetto v4.00*.

3 Specifica dei componenti

3.1 Metodo e formalismo di specifica

L'esposizione dell'architettura in dettaglio dell'applicazione è esposta di seguito seguendo un approccio top-down a livelli. Si descrive quindi l'architettura partendo dal generale esponendo inizialmente le componenti più teoriche: i package fino a quelle più concrete: le classi con i relativi metodi, attributi e relazioni di ereditarietà. Per distinguere in modo immediato le componenti di librerie dai componenti dell'applicativo si è deciso di associare ciascuna libreria ad un colore specifico:

- Android SDK: classi rappresentati in verde;
- JGraphT: classi rappresentate in grigio;
- AltBeacon: classi rappresentate in arancione;
- Java API: classi rappresentate in azzurro.

Mentre le classi dell'applicativo sono rappresentate nel classico giallo.

Per ogni package si specifica:

- il nome;
- una descrizione;
- il package da cui discende;
- le interazioni con gli altri package;
- gli eventuali package contenuti;
- le classi contenute affiancate da un riferimento alla descrizione completa.

Per ogni classe si specifica:

- il nome;
- il tipo;
- l'eventuale classe che estende;
- le eventuali interfacce che implementa;
- la visibilità;

- una descrizione;
- la lista dettagliata degli attributi;
- la lista dettagliata dei metodi.

Per i diagrammi dei package e delle classi si utilizza il formalismo *UML 2.0*.

4 Schema base di dati

Di seguito viene presentata lo schema in UML della base di dati implementata nell'applicativo con SQLite e gestito dal componente `DataManager` e implementata nel server remoto. Lo schema illustra le relazioni tra le entità che costituiscono il grafo rappresentate l'edificio di interesse. Si fa notare che la base di dati non memorizza separatamente gli elementi che compongono i grafi.

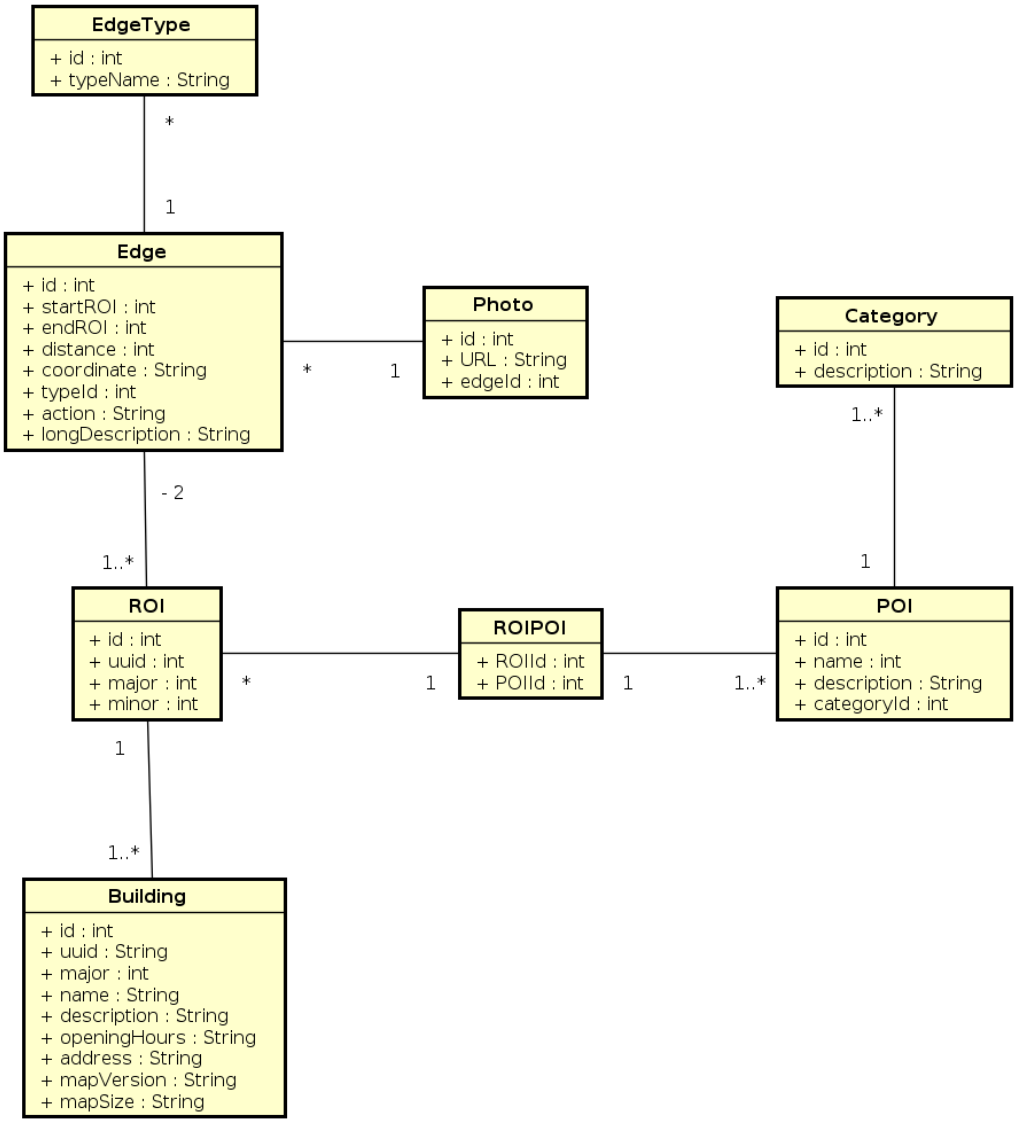

Figura 1: Schema UML - base di dati

5 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.

5.1 Avvio Service per il rilevamento beacon

Il diagramma in figura 2 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$_g$ `BeaconManagerAdapter` viene chiamato il metodo `onCreate()` nel quale viene creata un istanza della classe `BeaconManager` offerta dalla libreria AltBeacon$_g$. 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.

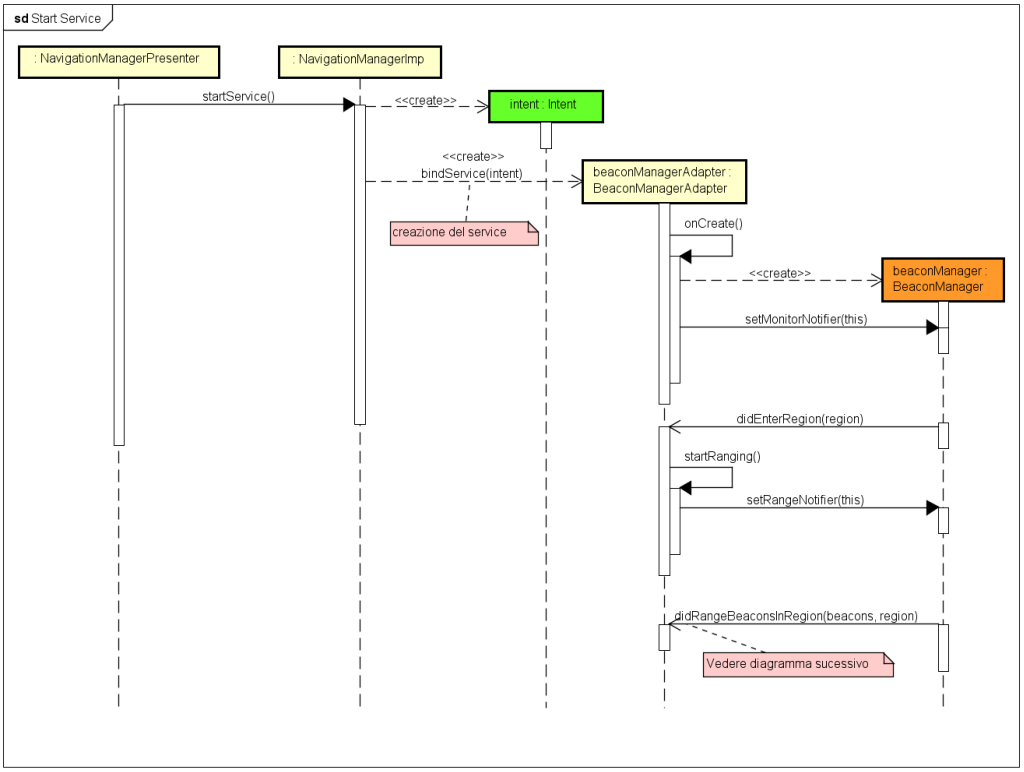

Figura 2: Diagramma di sequenza - Avvio di un service$_g$ per il rilevamento beacon

5.2 Elaborazione beacon rilevati e comunicazione broadcast

Il diagramma in figura 3 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$_g$ e implementa il listener di `BeaconManager`: `RangeNotifier` il quale scatenerà, dopo una scansione, l'evento `didRangBeaconsInRegion()` 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$_g$ (`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$_g$.

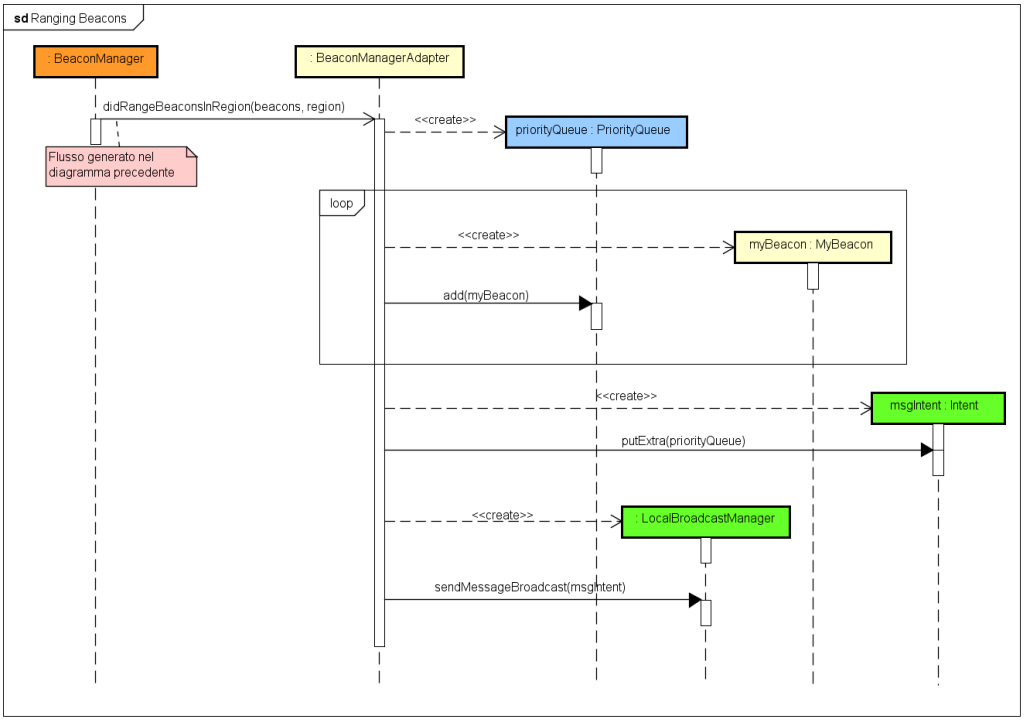

Figura 3: Diagramma di sequenza - Elaborazione beacon rilevati e comunicazione broadcast

5.3 Avvio navigazione

Il diagramma in figura 4 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 `NavigatiorManagerImp` passando come parametri la destinazione identificata dall'oggetto di tipo `PointOfInterest`. Il `NavigatorManagerImp` 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 scompatterà e le restituirà alla view e quindi all'utente.

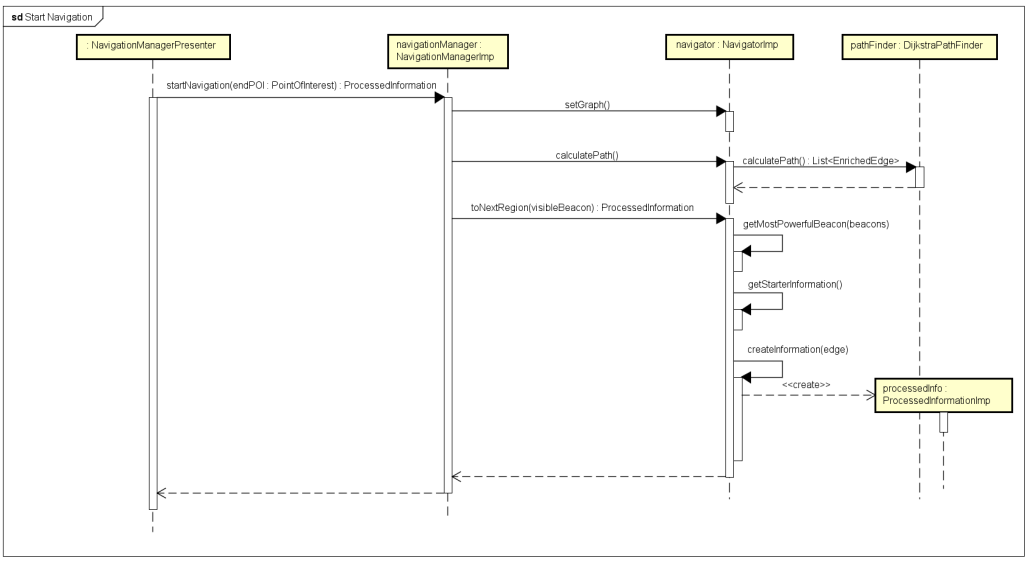

Figura 4: Diagramma di sequenza - Avvio navigazione

6 Tracciamento

6.1 Tracciamento Classi-Requisiti

6.2 Requisiti-Classi

A Diagrammi riassuntivi package significativi

Di seguito sono riportati i package più significativi dell'applicativo qualora non fosse completamente chiara la relazione tra le componenti e le classi al suo interno. Per chiarezza e spazio le classi rappresentate all'interno dei package sono senza metodi e attributi.

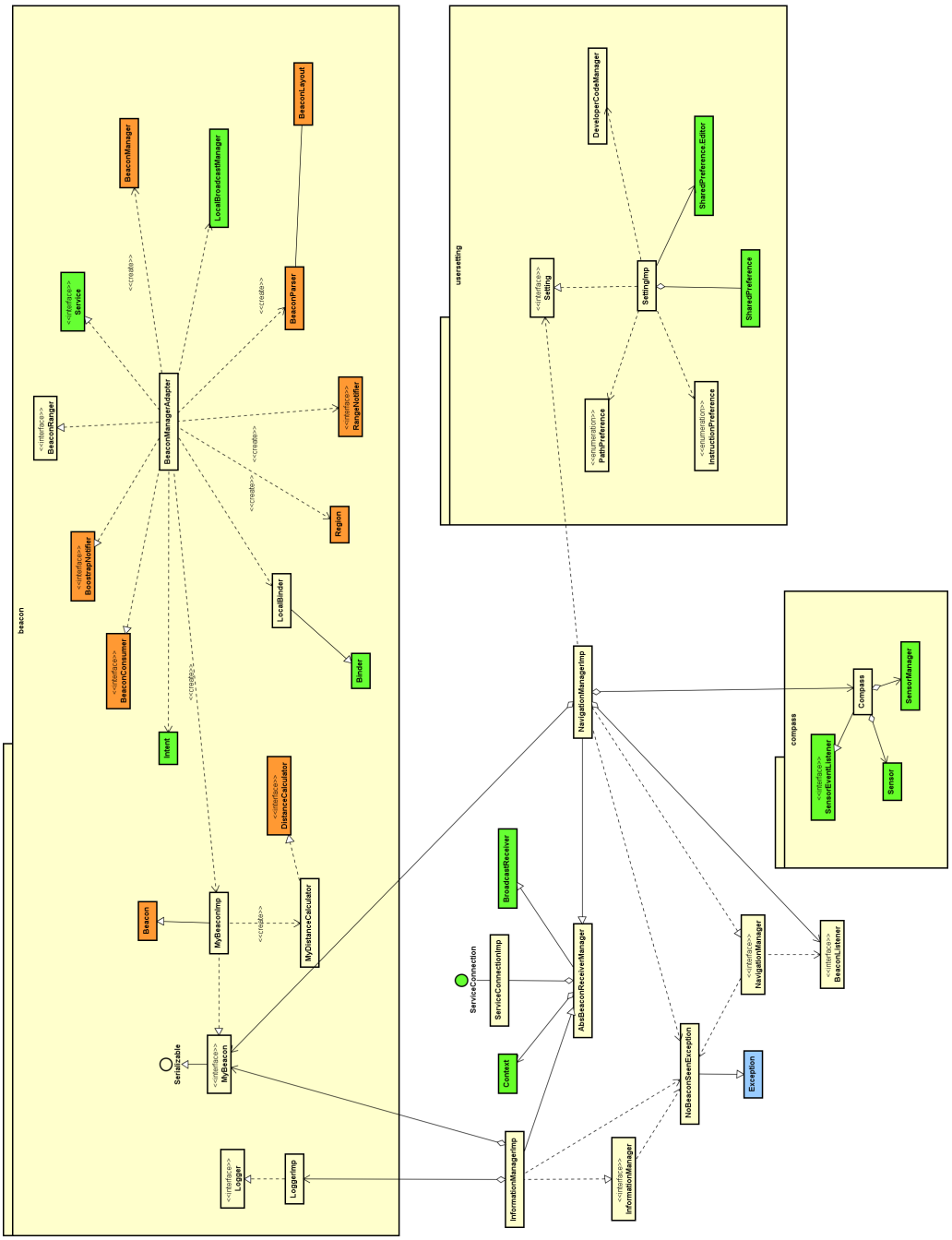

Figura 5: Diagramma delle classi - model

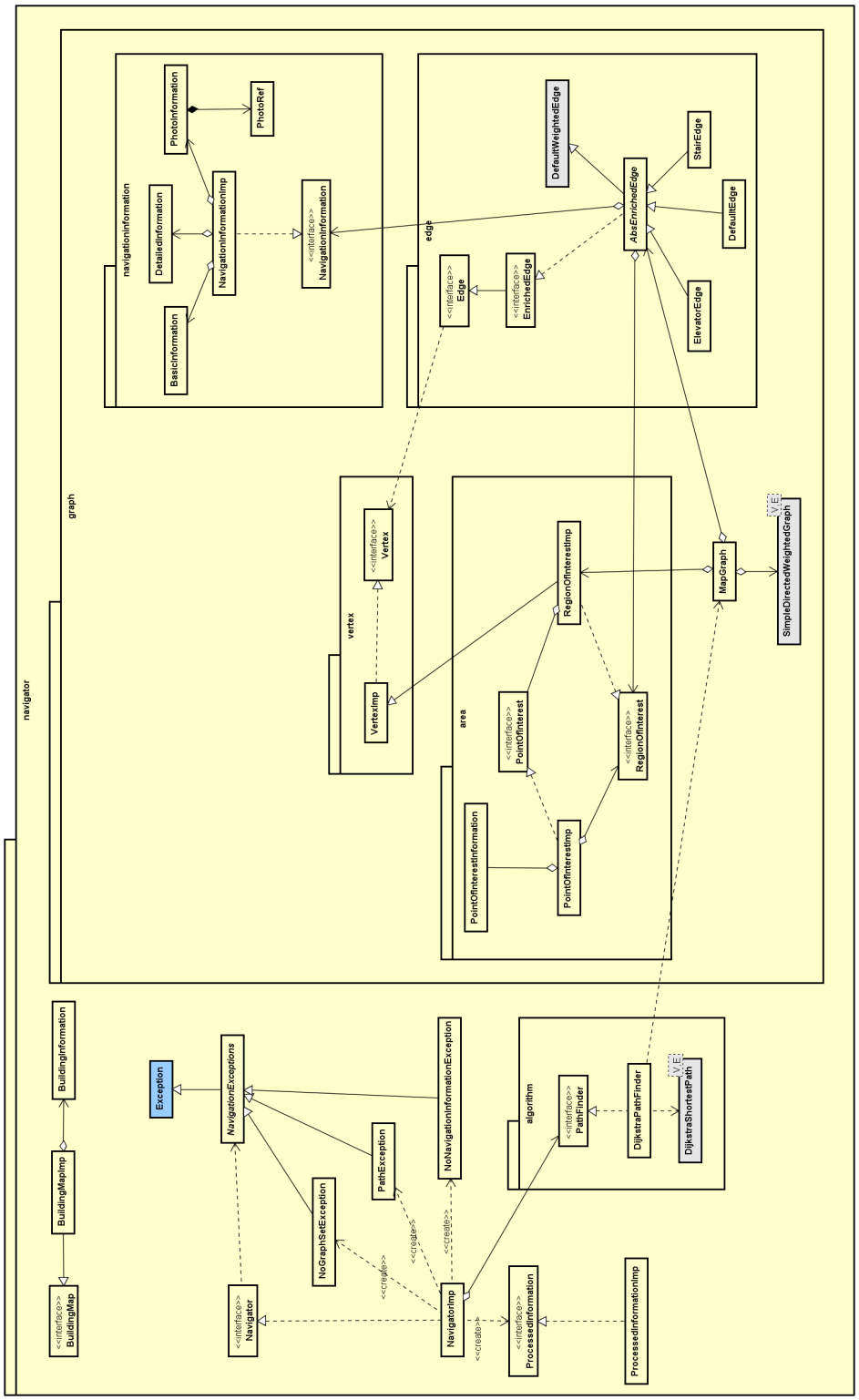

Figura 6: Diagramma delle classi - model::navigator 592.0pt

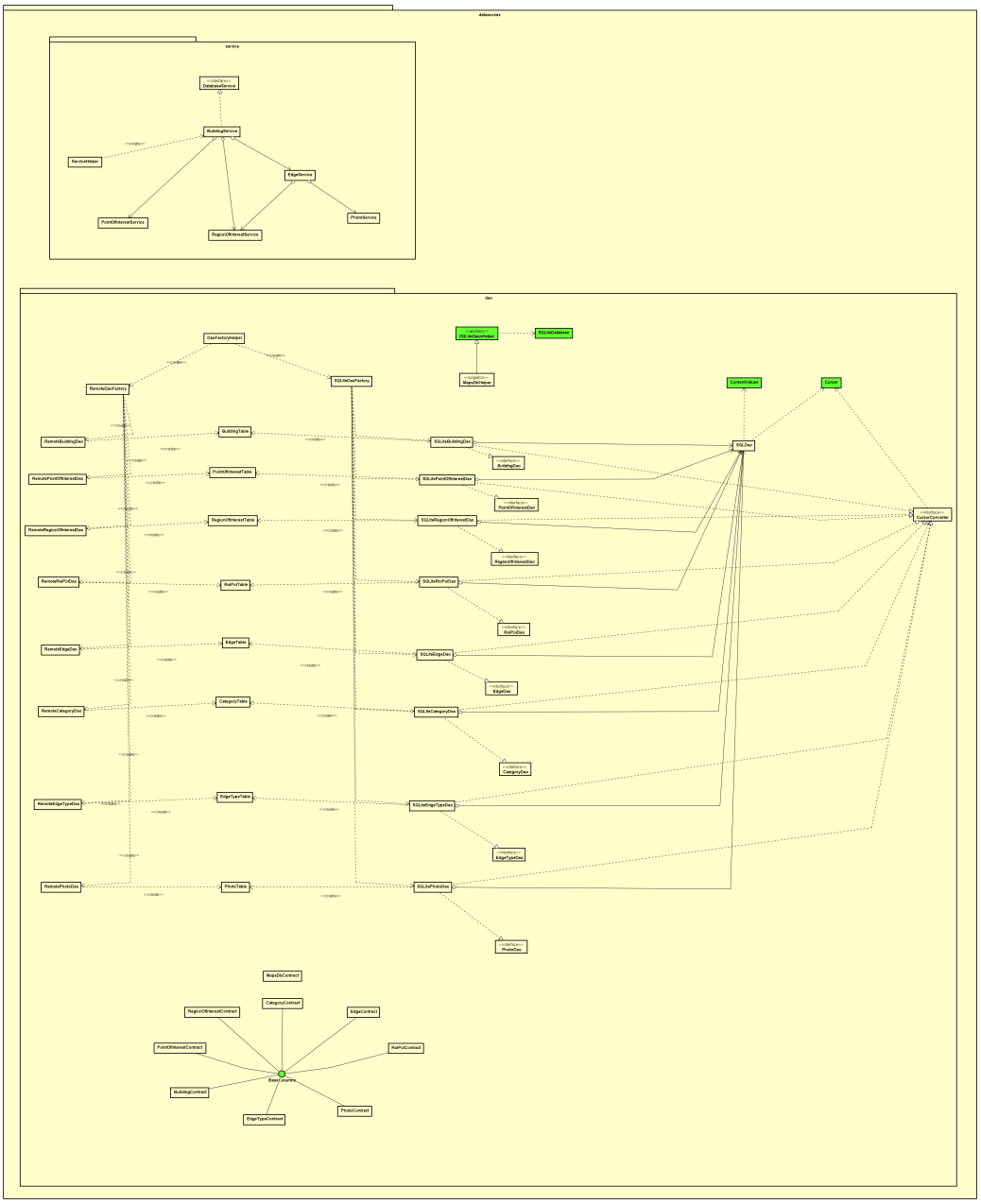

Figura 7: Diagramma delle classi - model::dataaccess