An Android application using a modular architecture for indoor localization

Michele Agostini

26th April 2017

Poor indoor localization.

Beautiful is the new *useful*

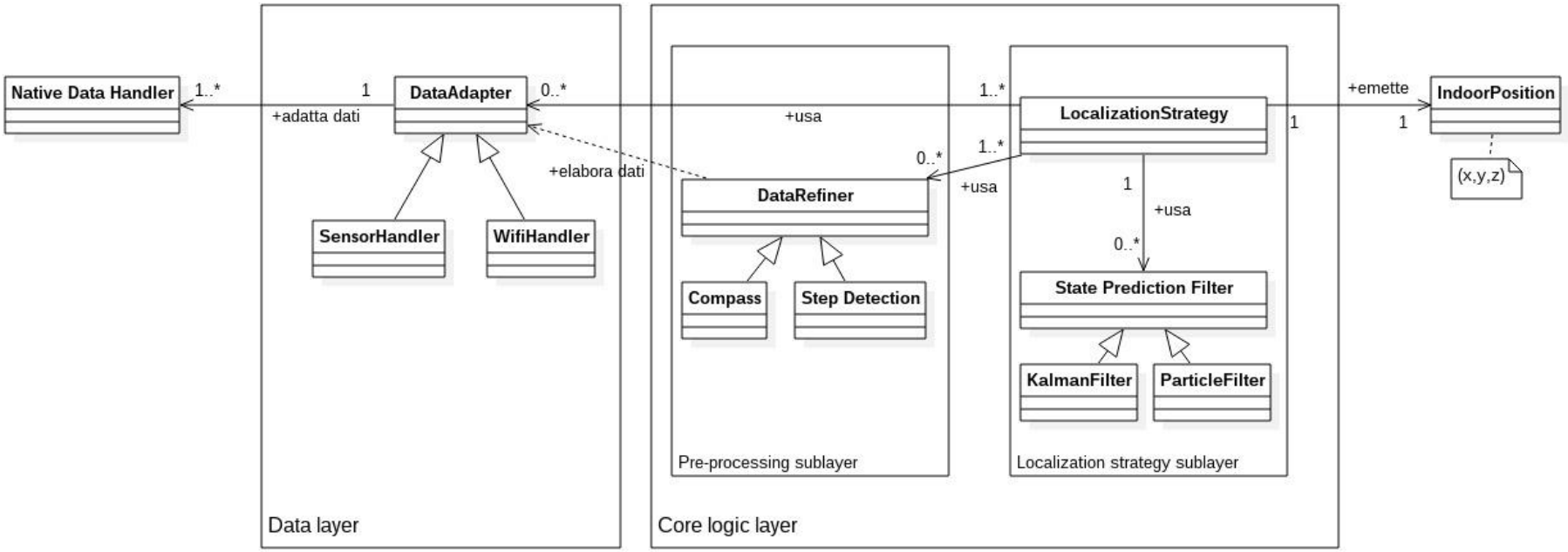
Benefits

- Modular
  - Maintainable
  - Reusable
  - Extensible
- Portable

Drawbacks

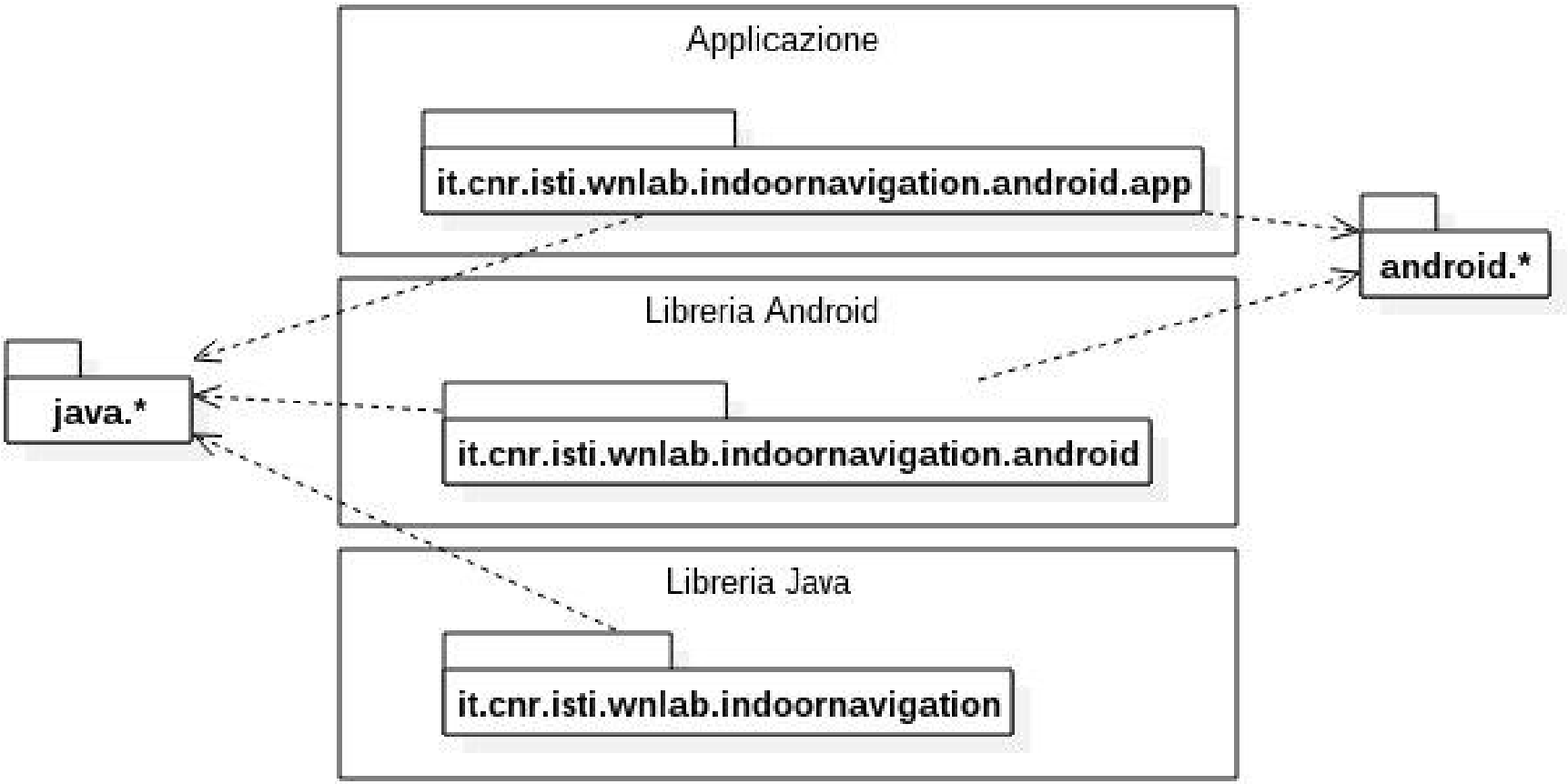
- Architecture's overhead
- Non-native code

Modularity: components in layers (conceptual)

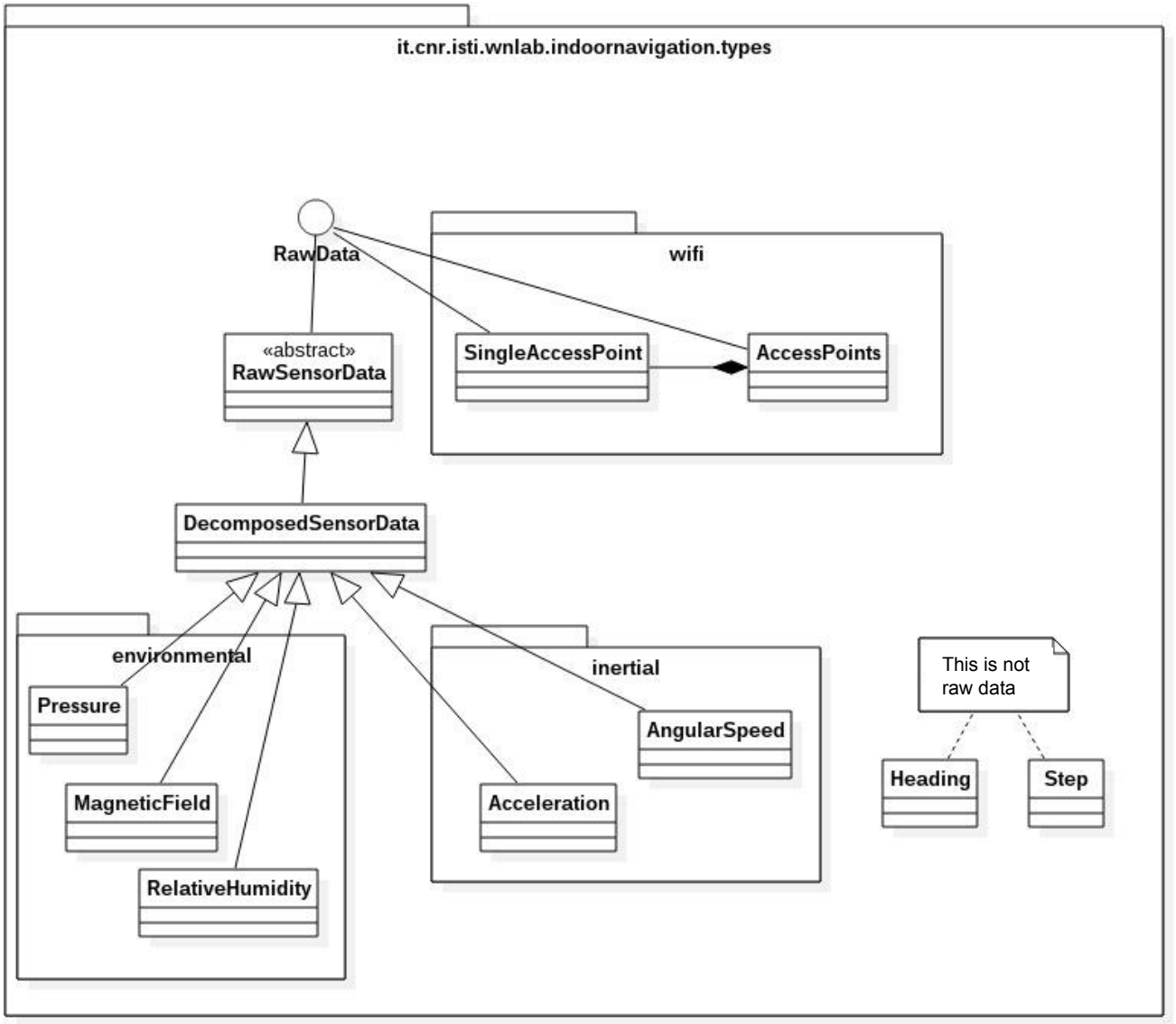


data flow

Portability: a layered architecture



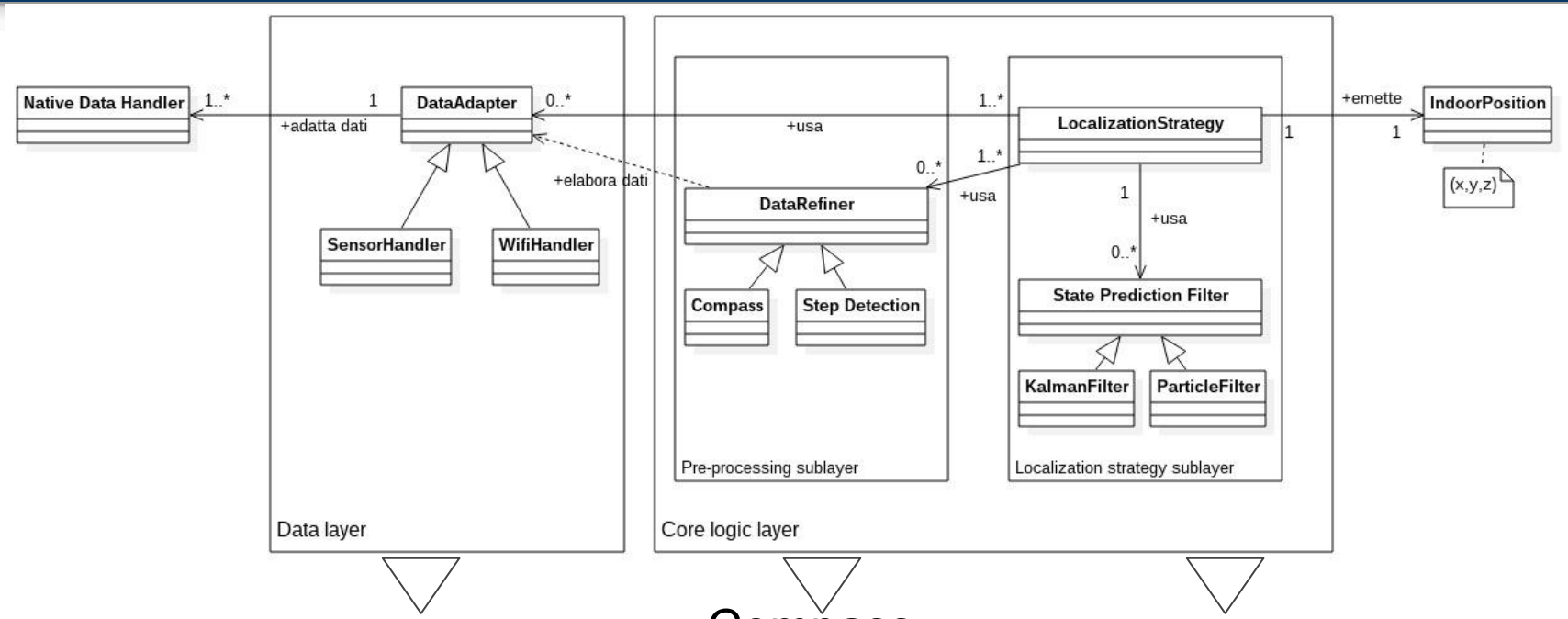
Portability: system-independence through type abstractions



Clean code

(more or less)

The concept becomes reality



- IndoorMap
- Emitter
- DataEmitter

Android:

- SensorData Emitter
- WifiScanner

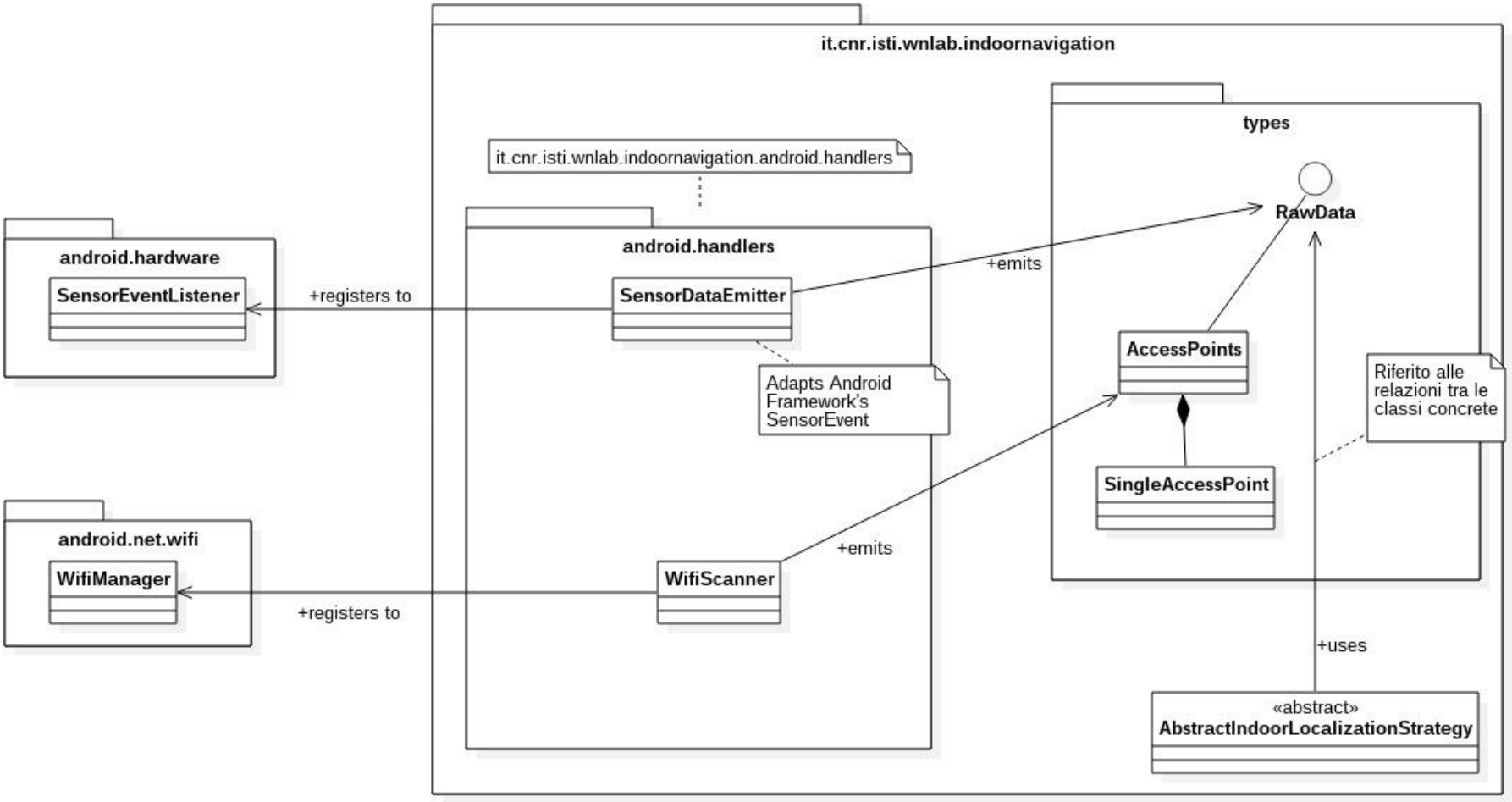
- Compass
- StepDetection
- PDR
- FingerprintMap
- DistancesMap

Android:

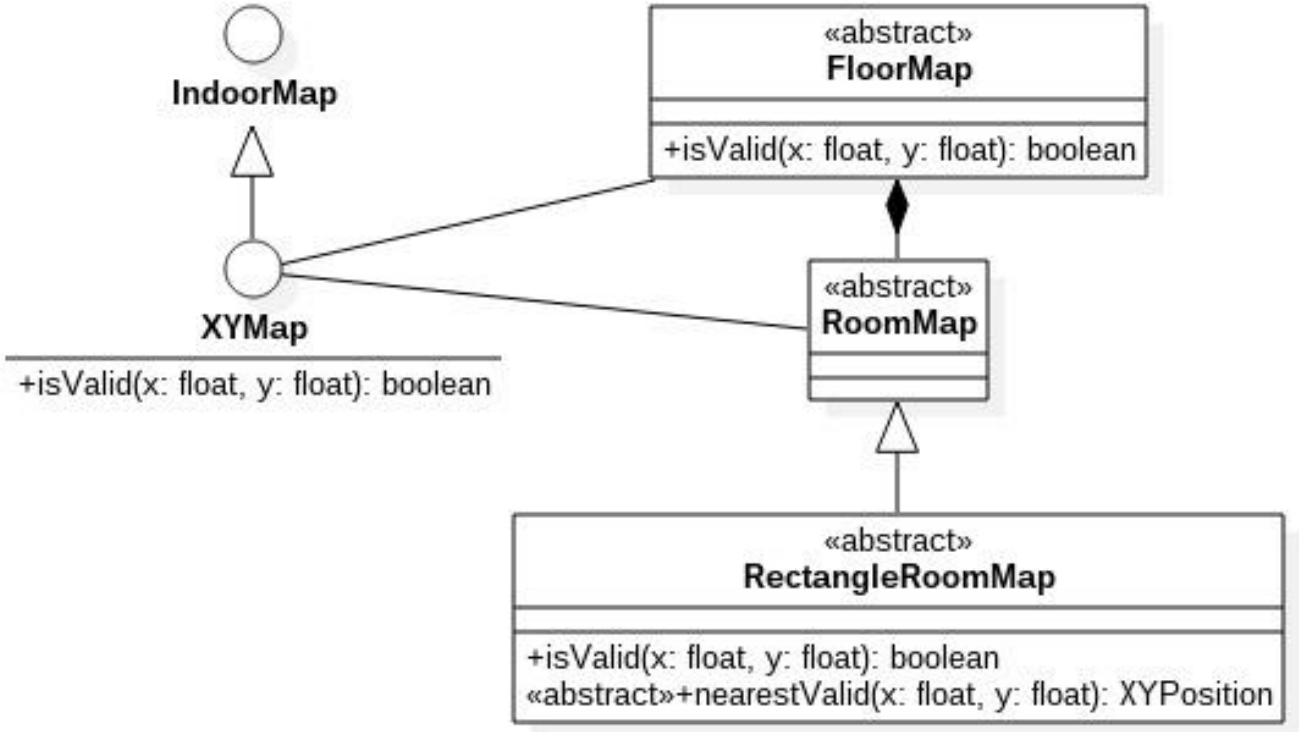
- SD and Compass implementations*

- Localization Strategy
- Filter

Data Layer (with Android): Data handlers

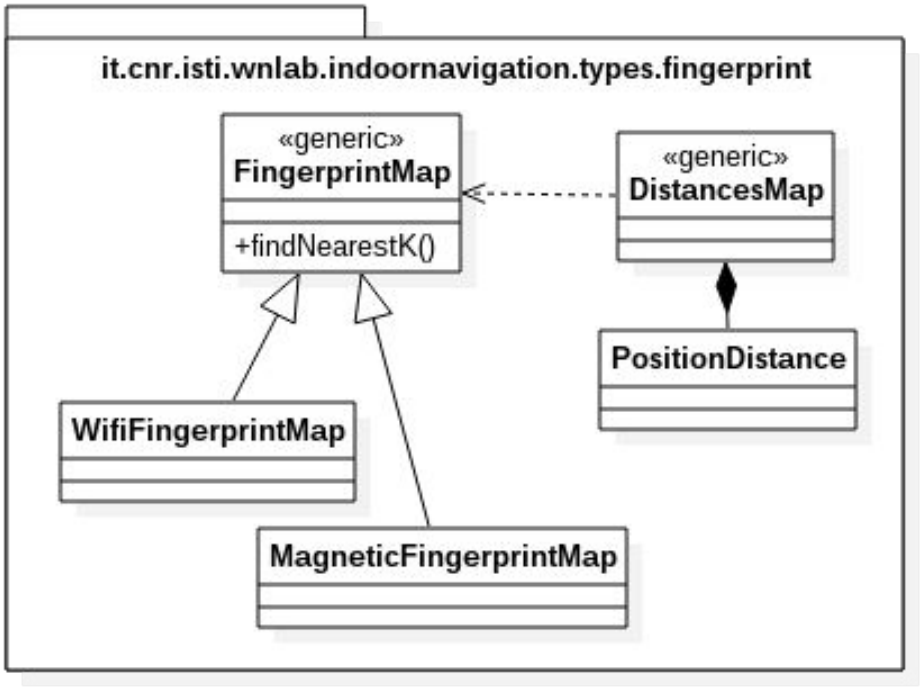


Data layer: IndoorMap



- Abstracts the area
- Responsible of loading geographical map data (if any, *wherever* it is)
- Exposes map-related operations

Data layer: FingerprintMap
Pre-processing layer: DistancesMap



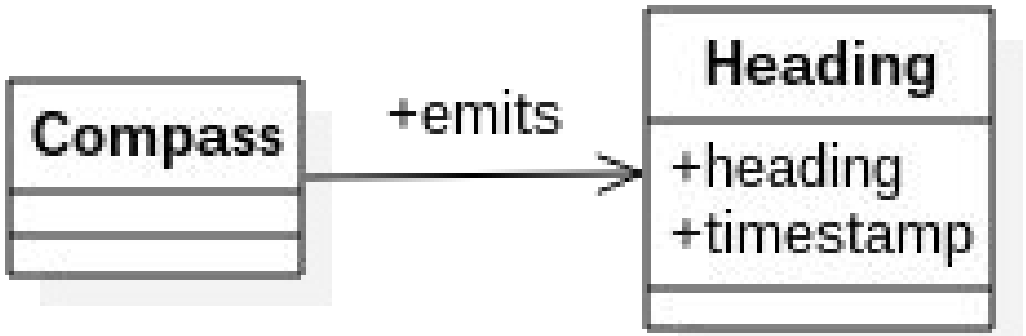
`class DistancesMap`

- Registers to a DataHandler
- Updates entry distances for each position in the database, for every update (ideally very slow
  => lazy policy)

`class FingerprintMap`

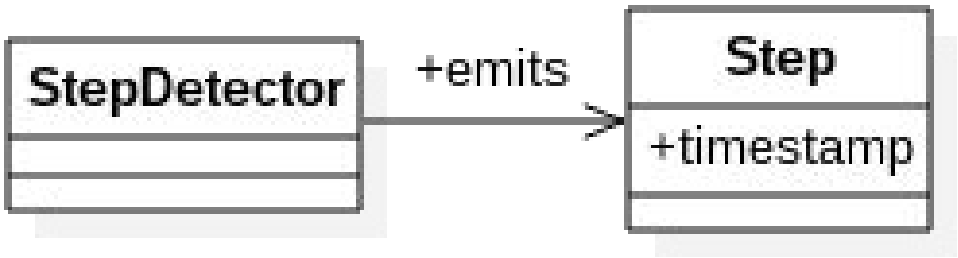
- Loads fingerprint database
- Expose useful fingerprint-related operations

Pre-processing layer: A heading emitter (commonly named Compass)



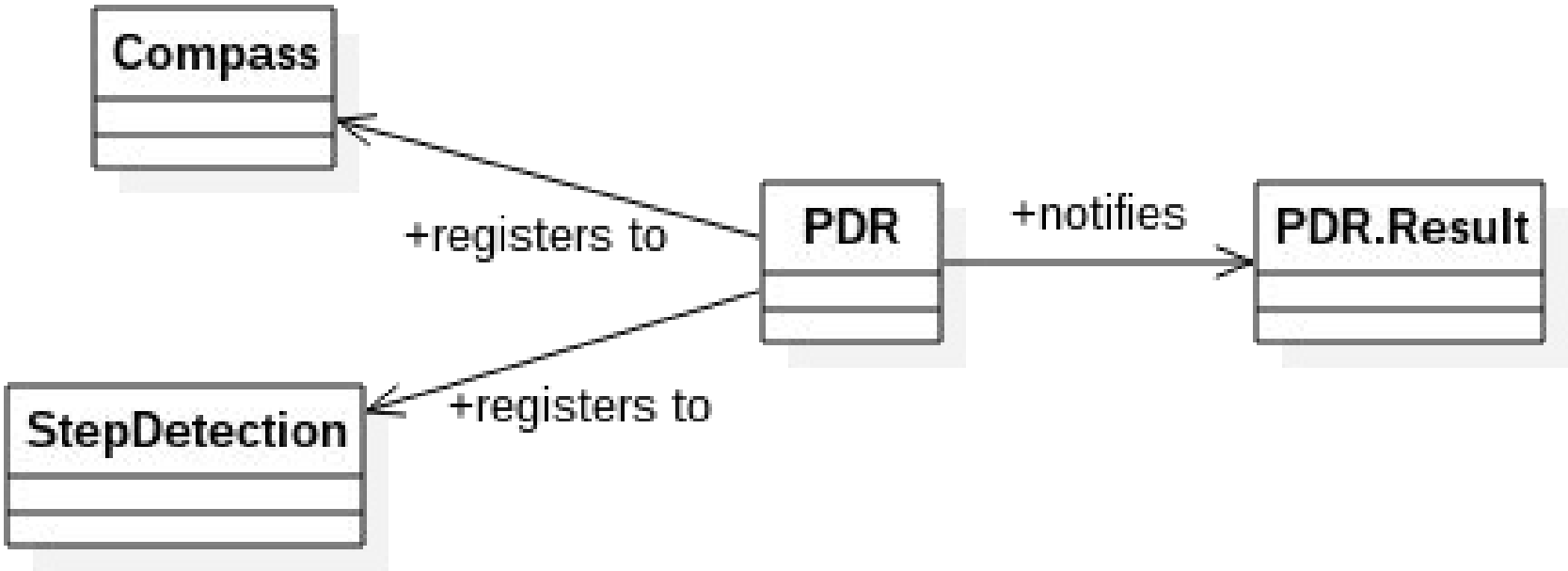
- `public abstract class Compass extends AbstractEmitter<Heading>;`
- The implementation is in the “Android” layer for simplicity

Pre-processing layer: A step emitter (commonly named StepDetector)



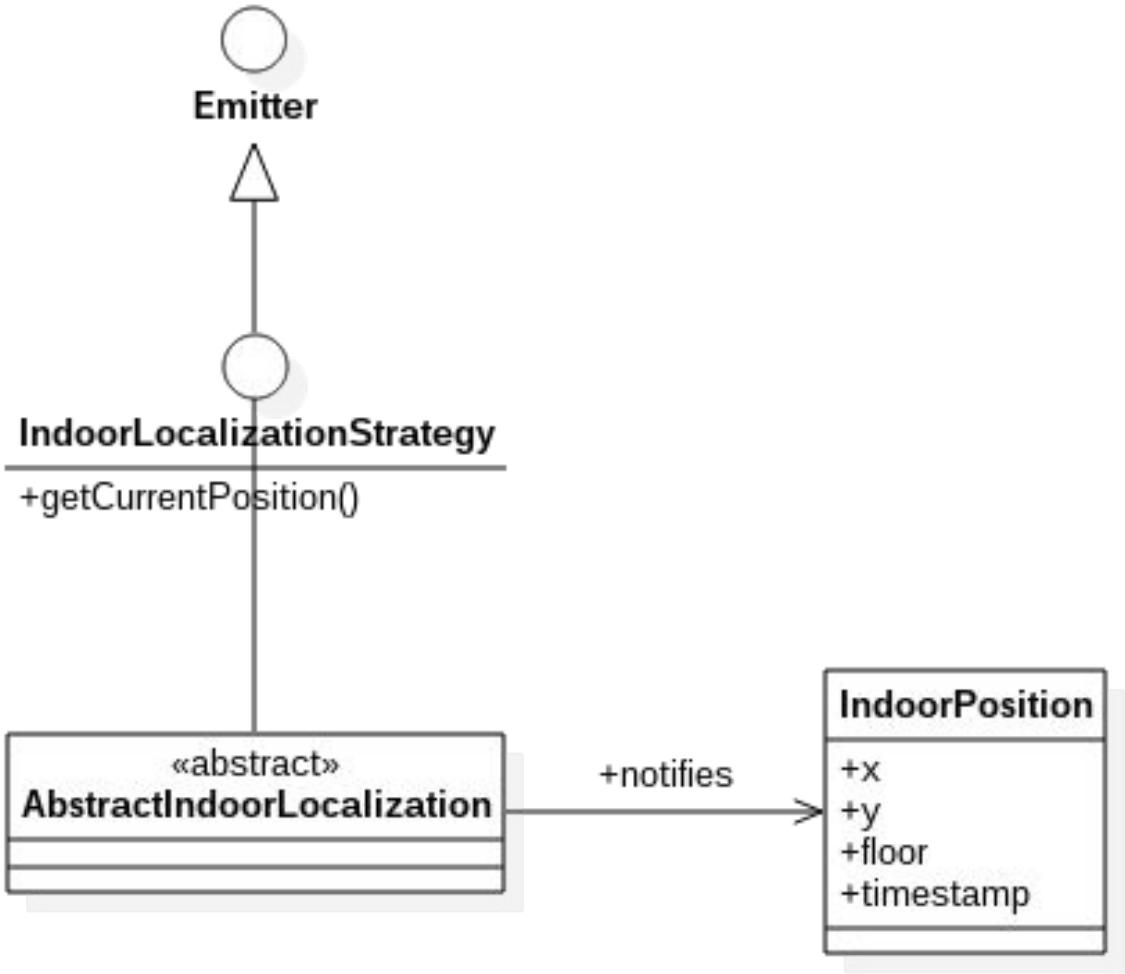
- `public abstract class StepDetector extends AbstractEmitter<Step>;`
- The implementation is in the “Android” layer but it makes no much sense

Pre-processing layer: Pedestrian Dead Reckoning (PDR)



LocalizationStrategy: an IndoorPosition emitter

- **DEPENDS ON** pre-processing modules and data handlers (injected).
- **RESPONSIBLE** of its observations.
- **DOES NOT INITIALIZE** pre-processing modules and data handlers
- **extends** AbstractEmitter <IndoorPosition>

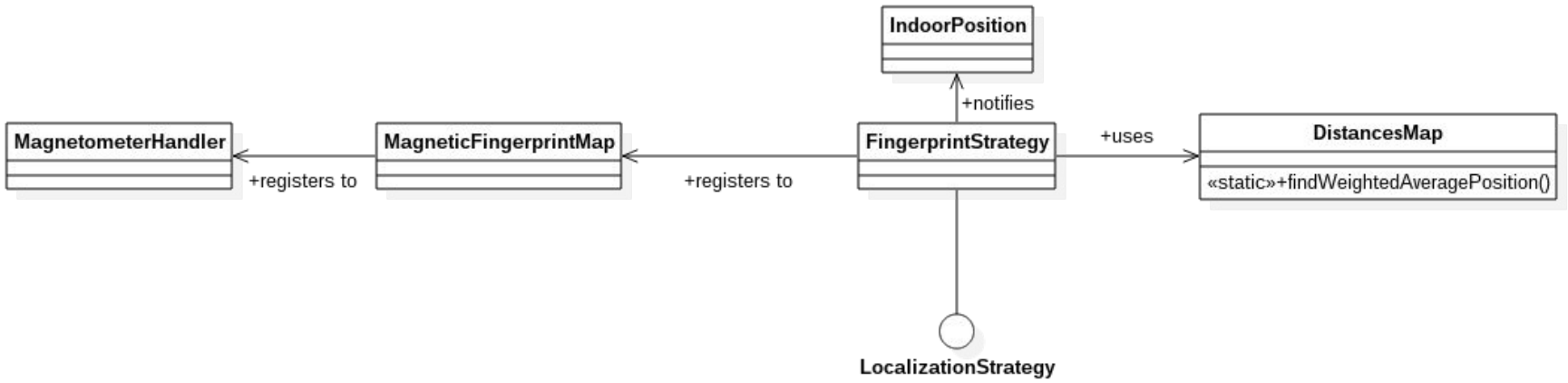


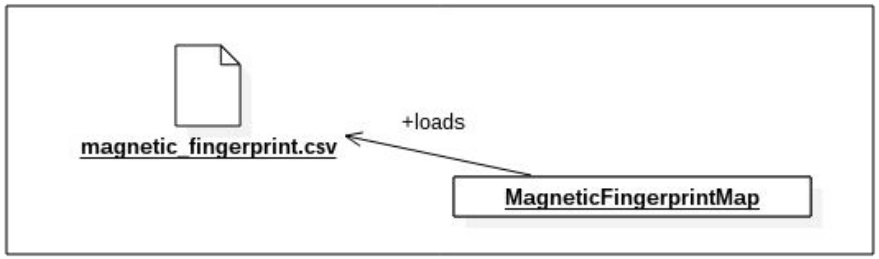
Internals: observers as ~~anonymous inner classes~~ λ-functions

```
public FixedStepPDR(
    Emitter<Heading> heading, Emitter<Step> stepDetector,
    float stepLength, float initialHeading) extends PDR {
    // Initialize step length and heading
    mStepLength = stepLength;
    mHeading = initialHeading;

    // Observer for heading changes
    mHeadingEmitter = heading;
    mHeadingObserver = (Observer) (data) -> {
        onHeadingChange(data.heading);
    };

    // Observer for step detection
    mStepDetector = stepDetector;
    mStepObserver = (Observer) (step) -> {
        onStep(step);
    };
}
```

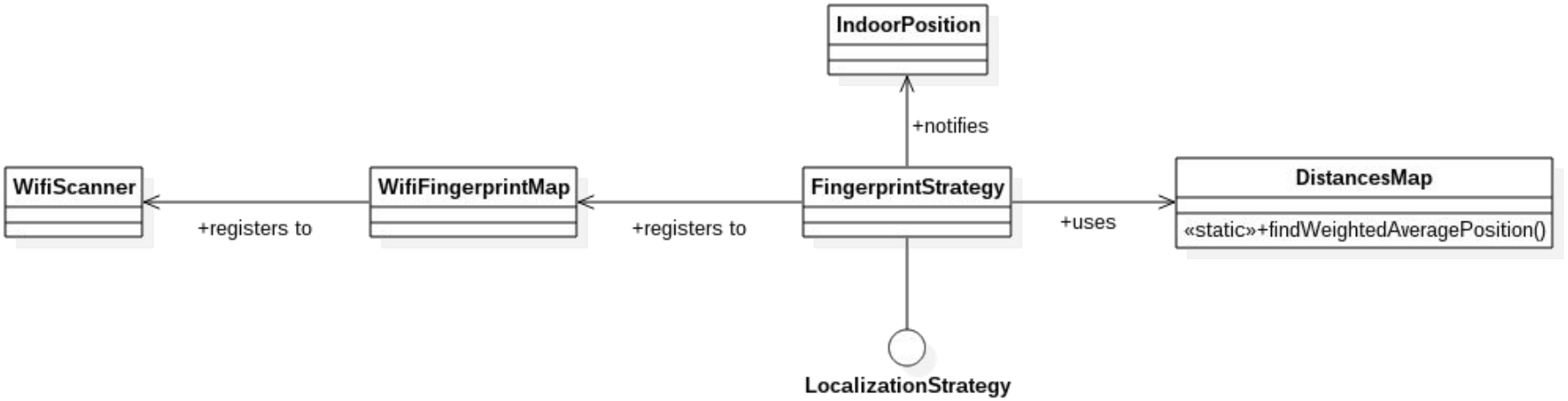
Time to play with LEGO

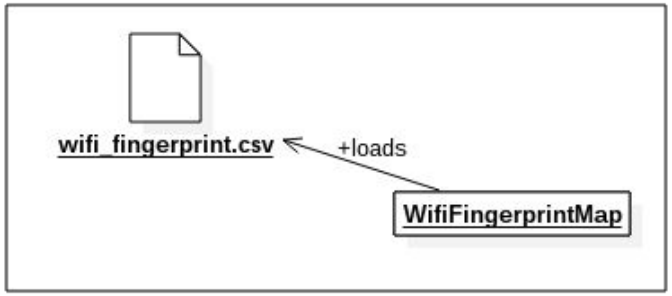
Dependencies: Magnetic Fingerprint-only localization strategy (in *utils* package)



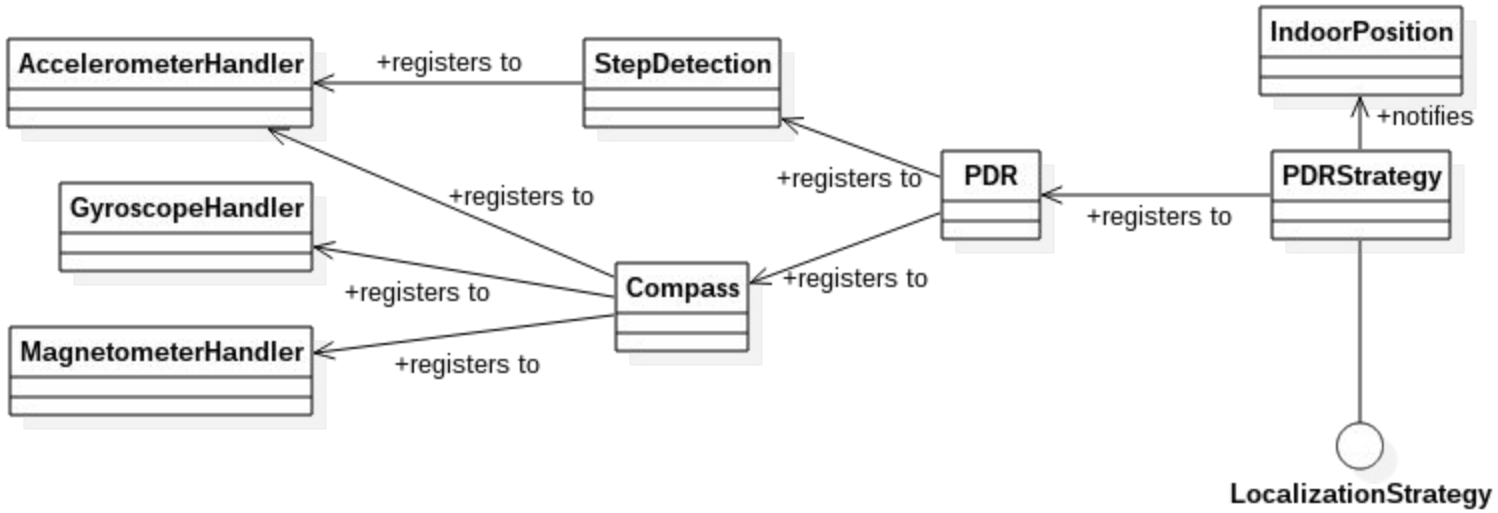


Dependencies: Wifi Fingerprint-only localization strategy (in *utils* package)





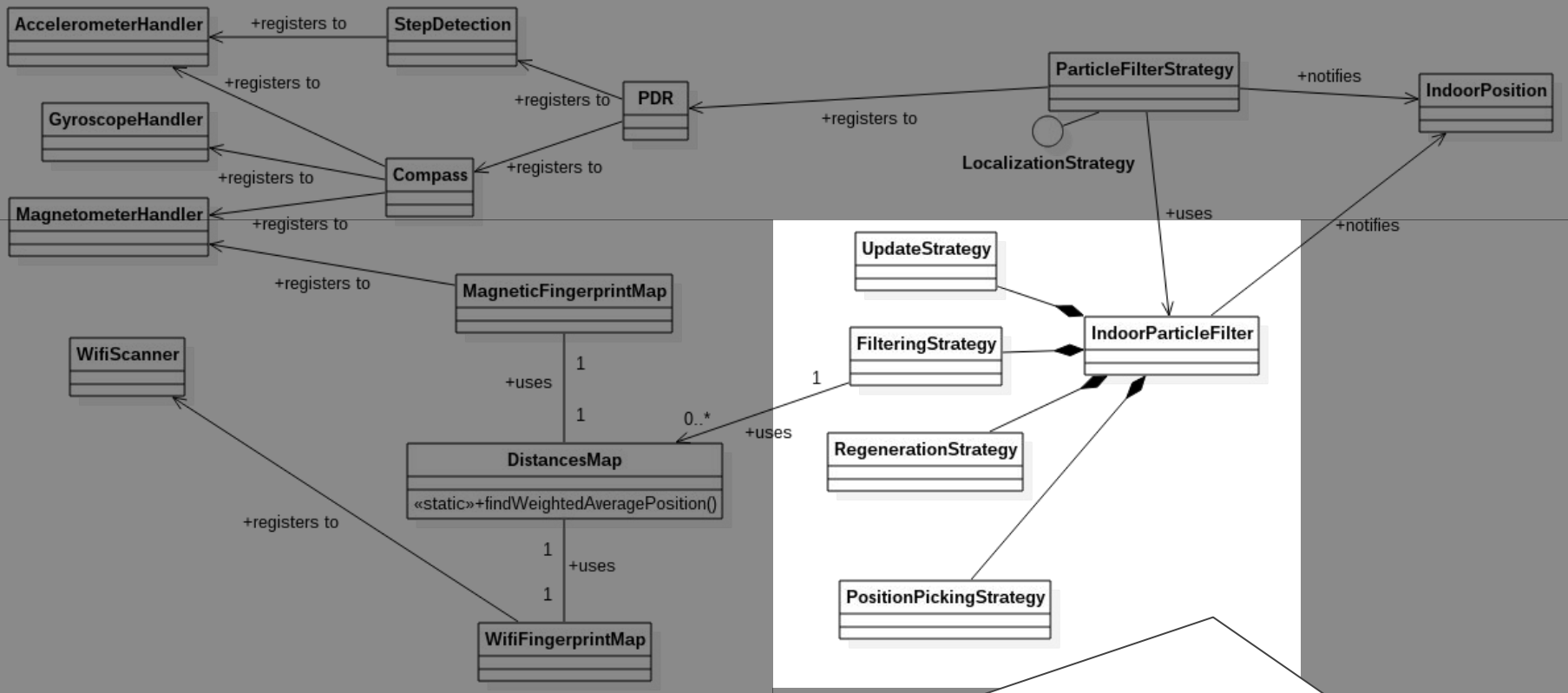
Dependencies: Localization only with PDR (in *utils* package)



A component in general:

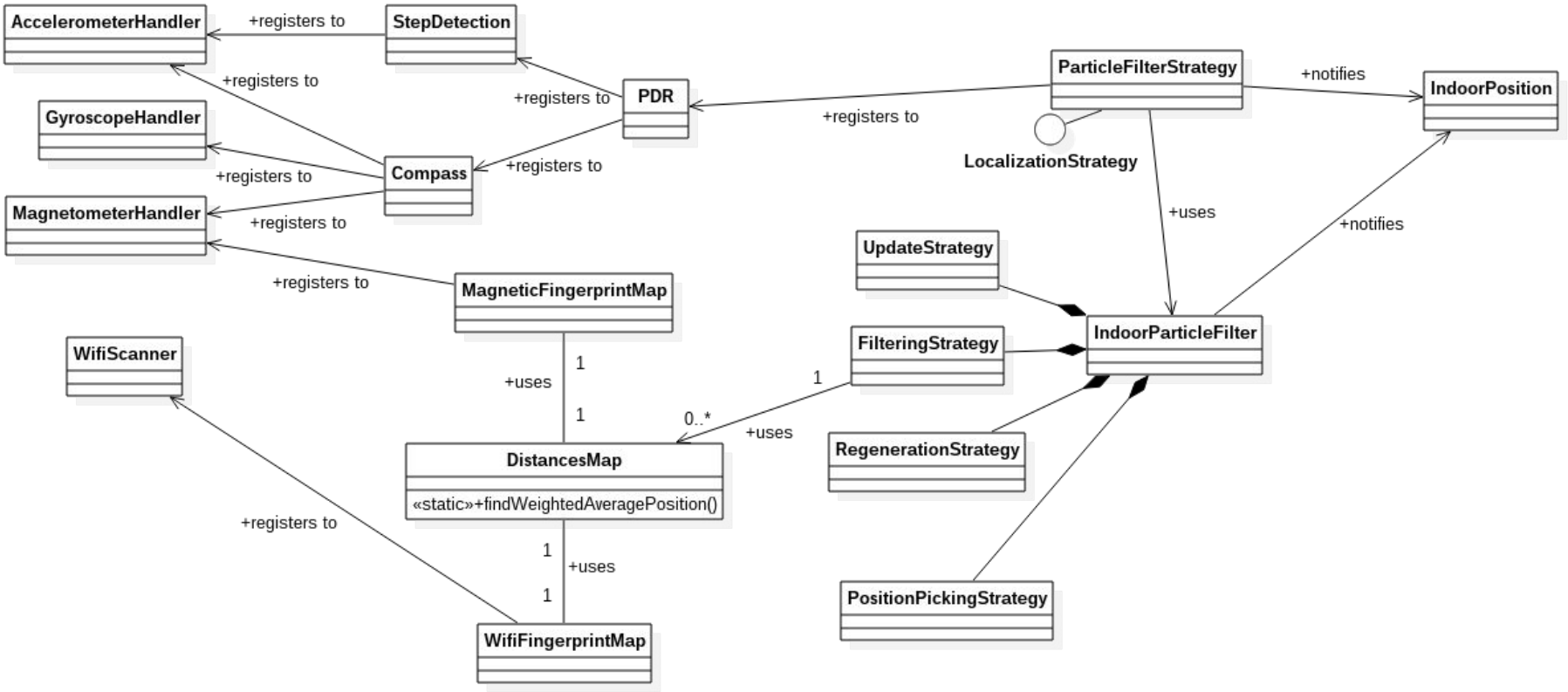
constructor injection, registration delegated to components

Dependencies: Localization strategy with Particle Filter (in *utils* package)

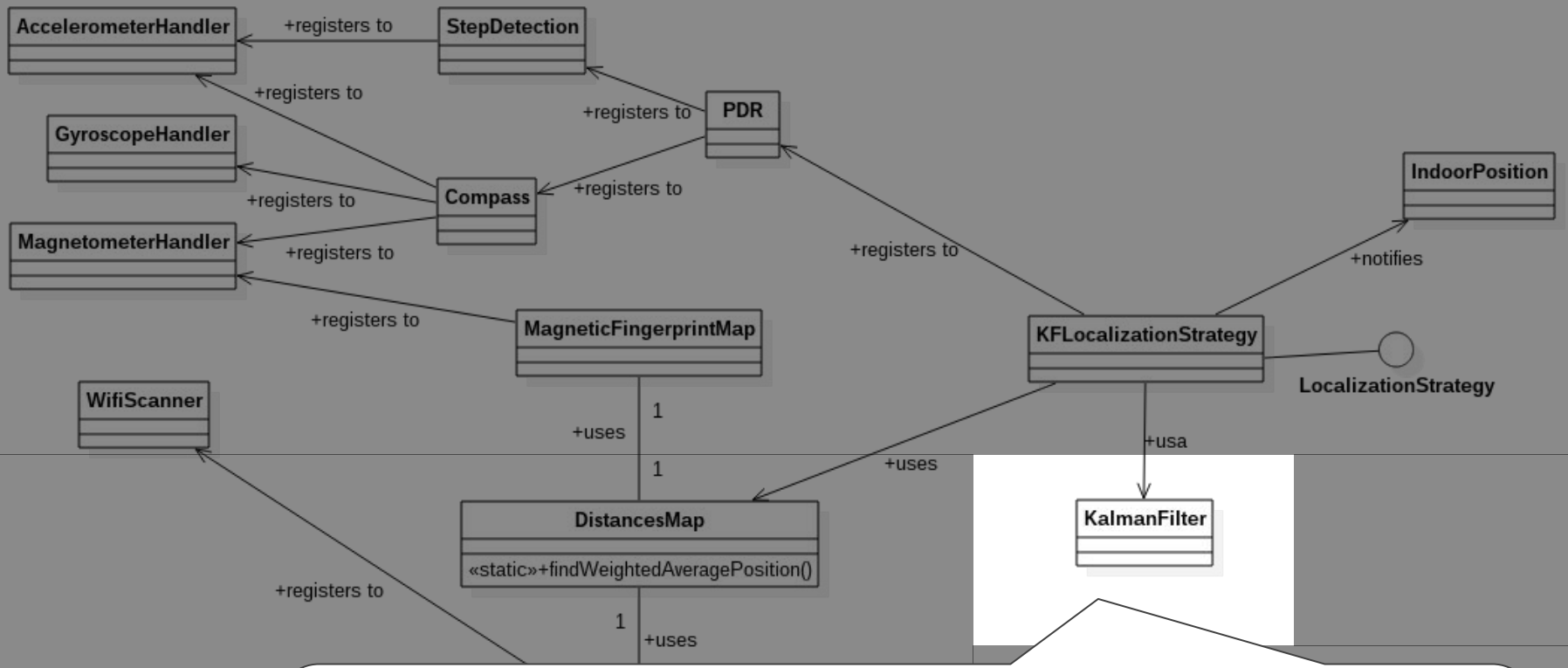


- *Composition over inheritance*
- Strategies operate on the same Collection<Particle>

Dependencies: Localization strategy with Particle Filter (in *utils* package)

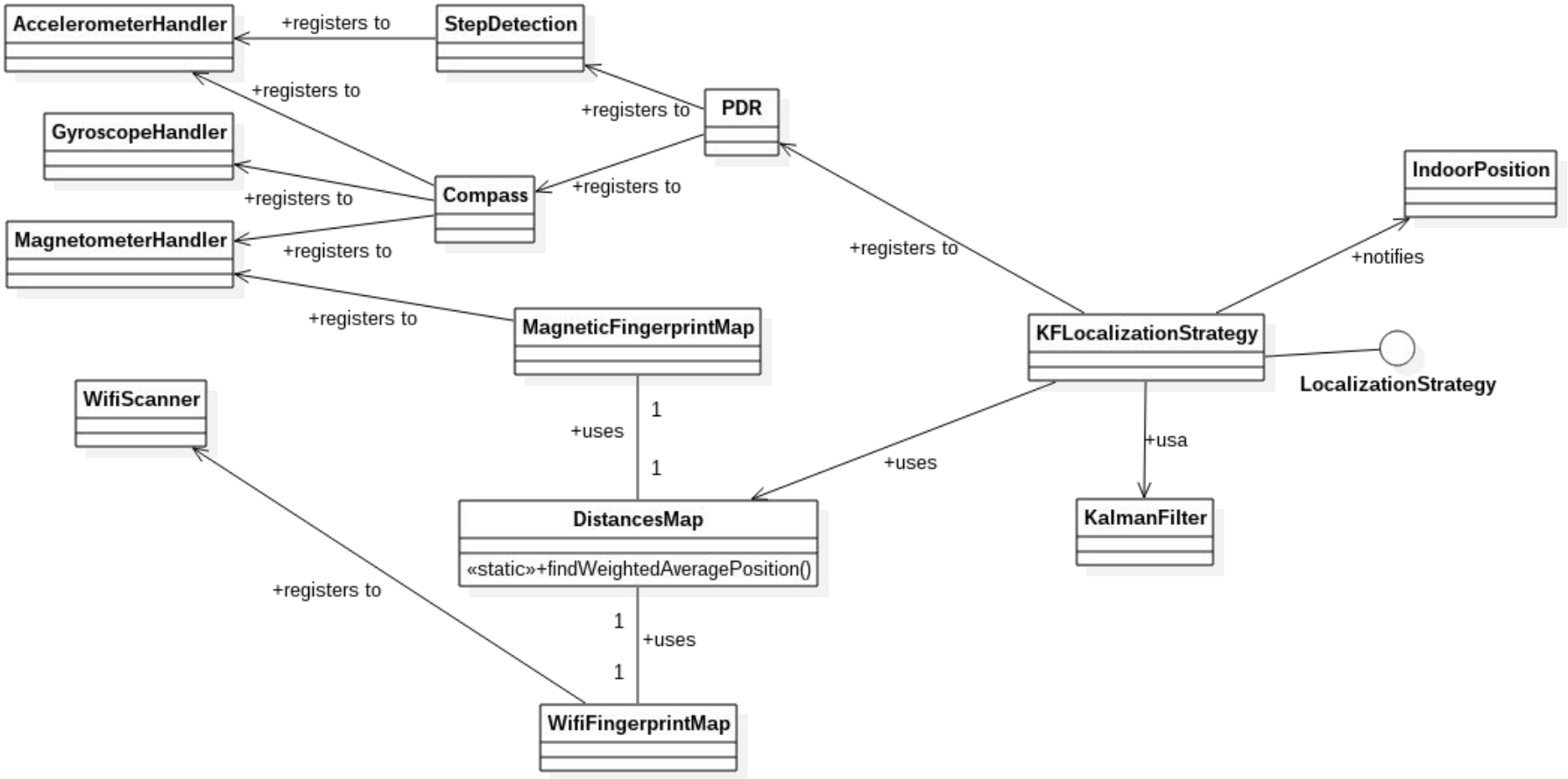


Dependencies: Localization strategy with Kalman Filter (in *utils* package)

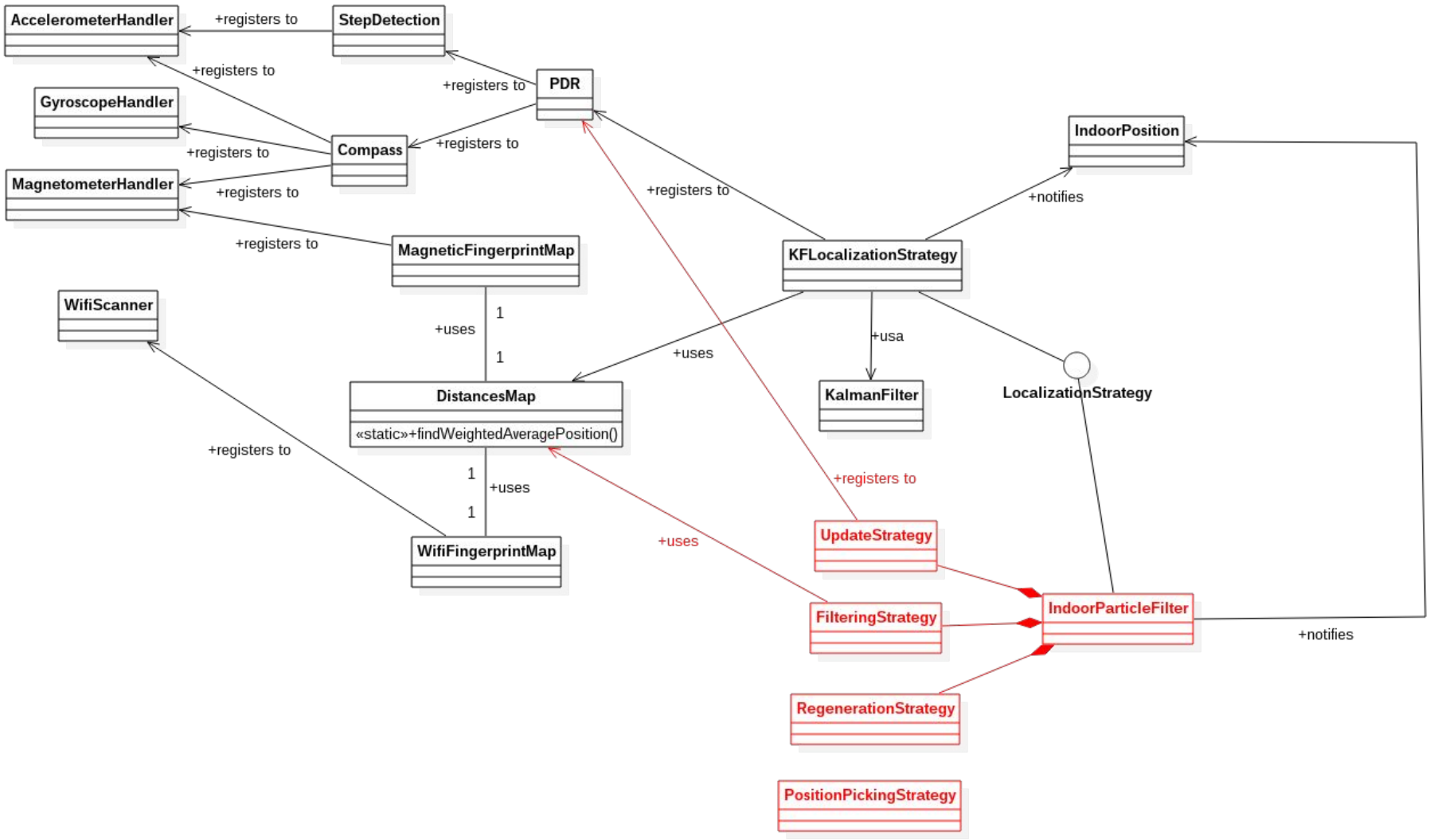


- An *abstract* class to extend (matrices must be consistent)

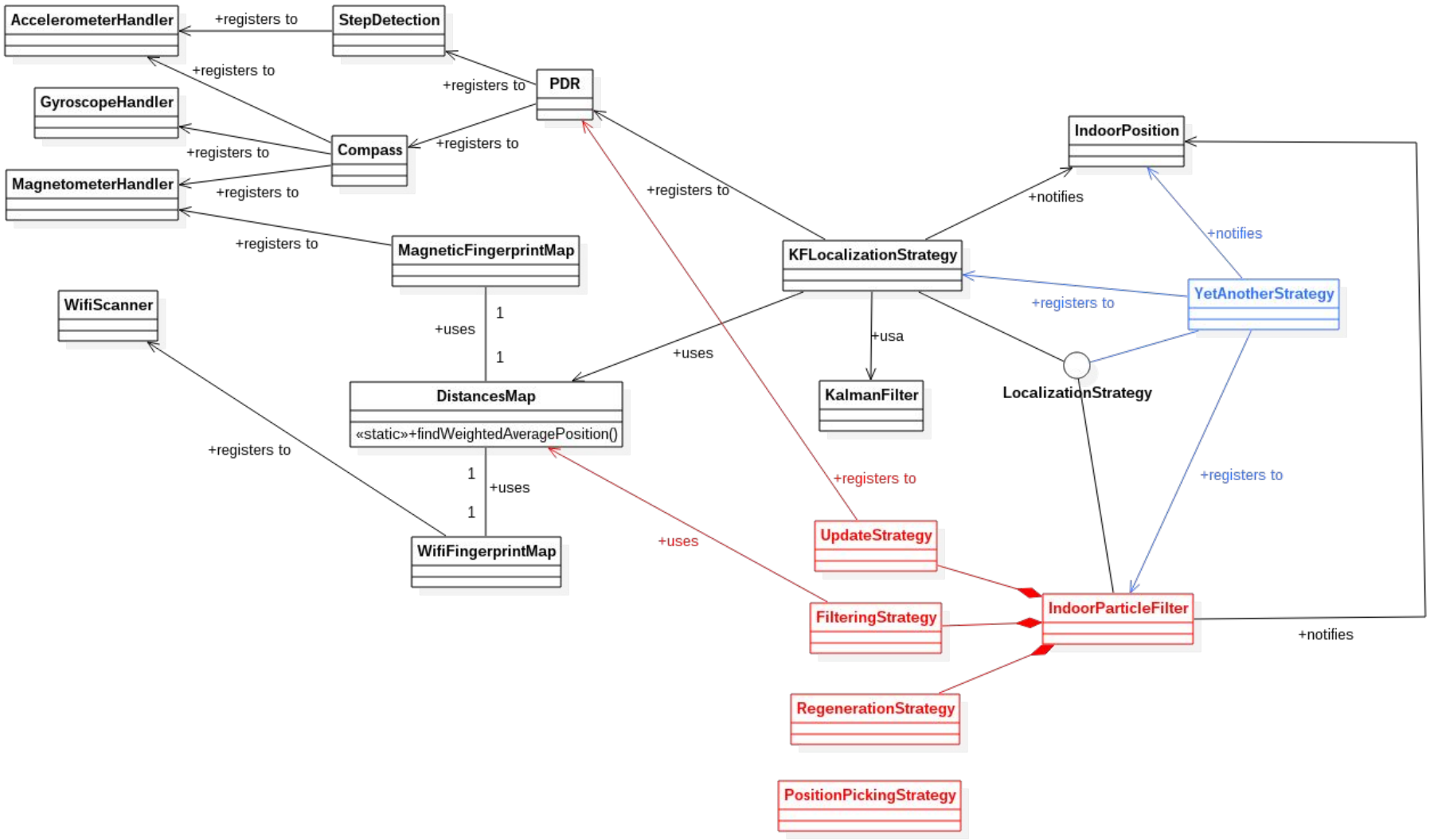
Dependencies: Localization strategy with Kalman Filter (in *utils* package)



I actually don’t know what it is, but you could also do this



I actually don’t know what it is, but you could also do this



</LEGO>

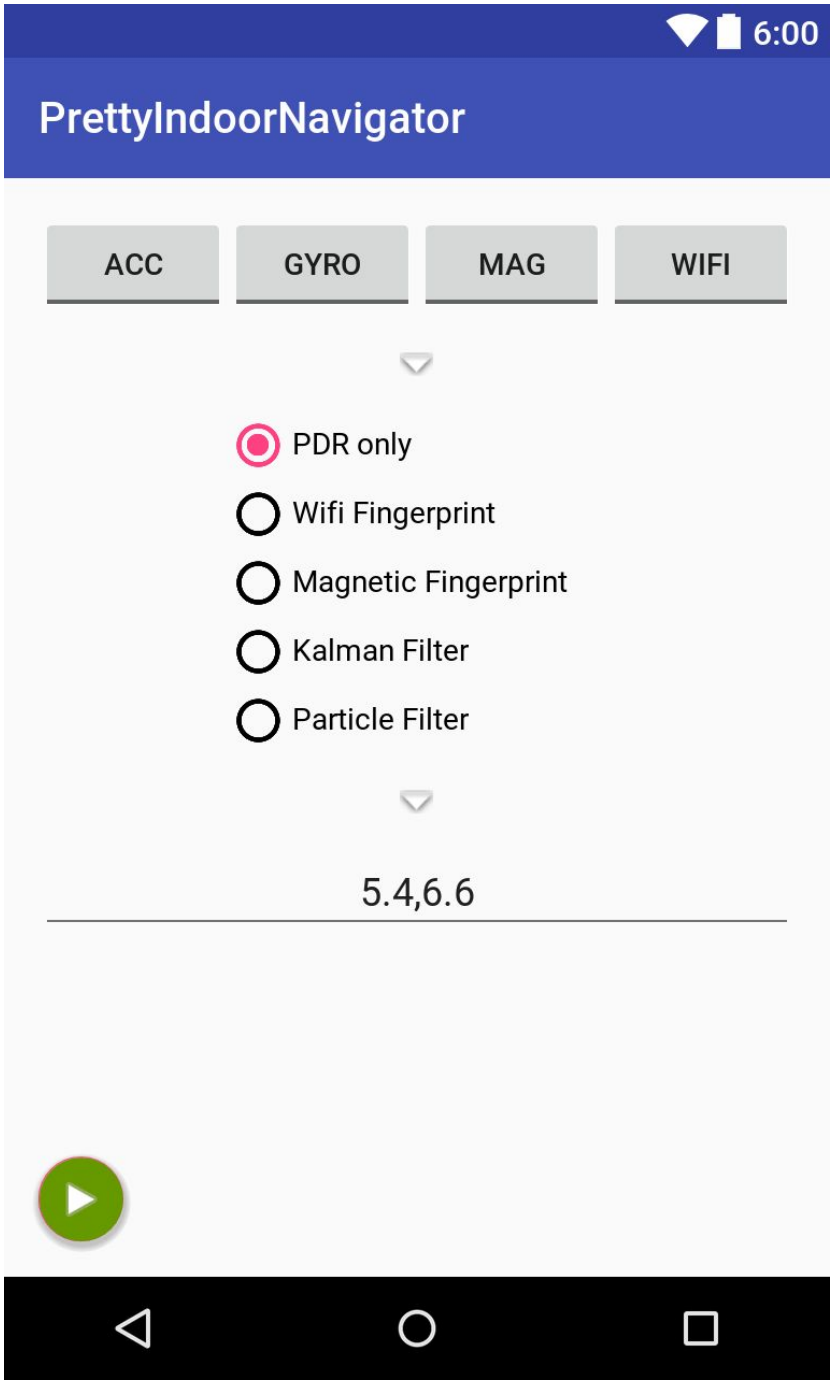
The Application

Use Cases:

- START/STOP LOCALIZING
  - LOGGING
  - TESTING
  - REUSABLE
- OFFLINE CONFIGURATION

Not implemented yet:

- ONLINE CONFIGURATION
  - ONLINE (DIS)ABLING EMITTERS
  - ONLINE STRATEGY CHOICE



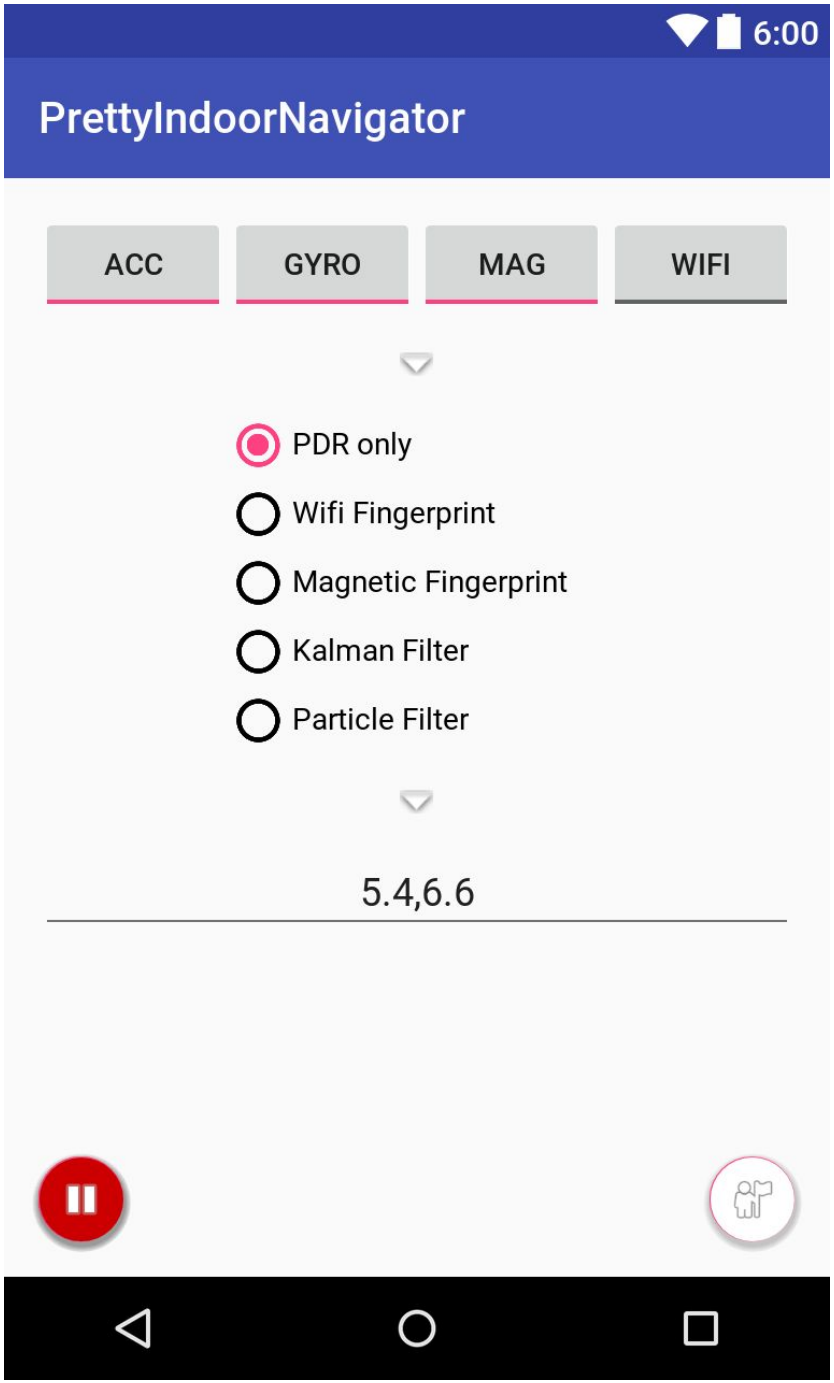
The Application

Use Cases:

- START/STOP LOCALIZING
  - LOGGING
  - TESTING
  - REUSABLE
- OFFLINE CONFIGURATION

Not implemented yet:

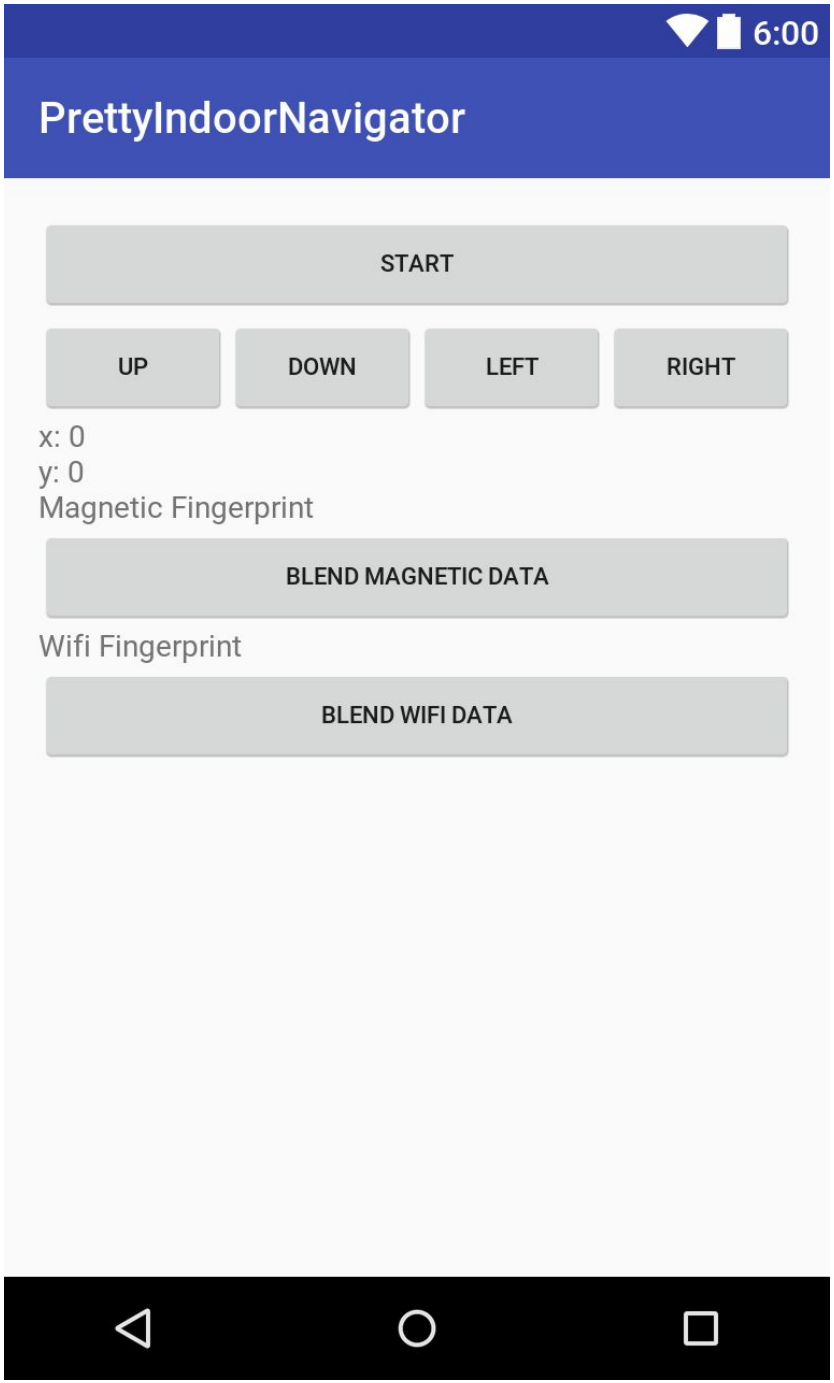
- ONLINE CONFIGURATION
  - ONLINE (DIS)ABLING EMITTERS
  - ONLINE STRATEGY CHOICE



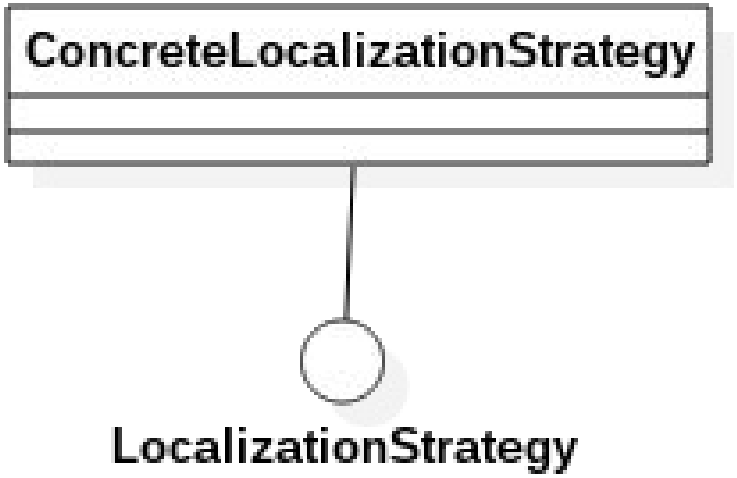
The Application (extra)

Use Cases:

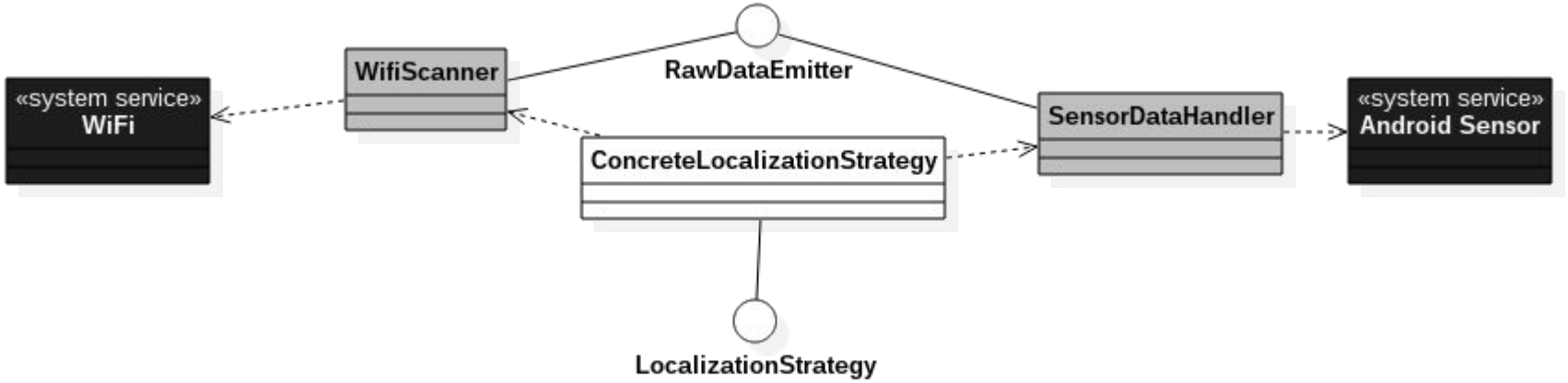
- FINGERPRINT ACQUISITION
- FINGERPRINT MAP'S CSV FILE CREATION



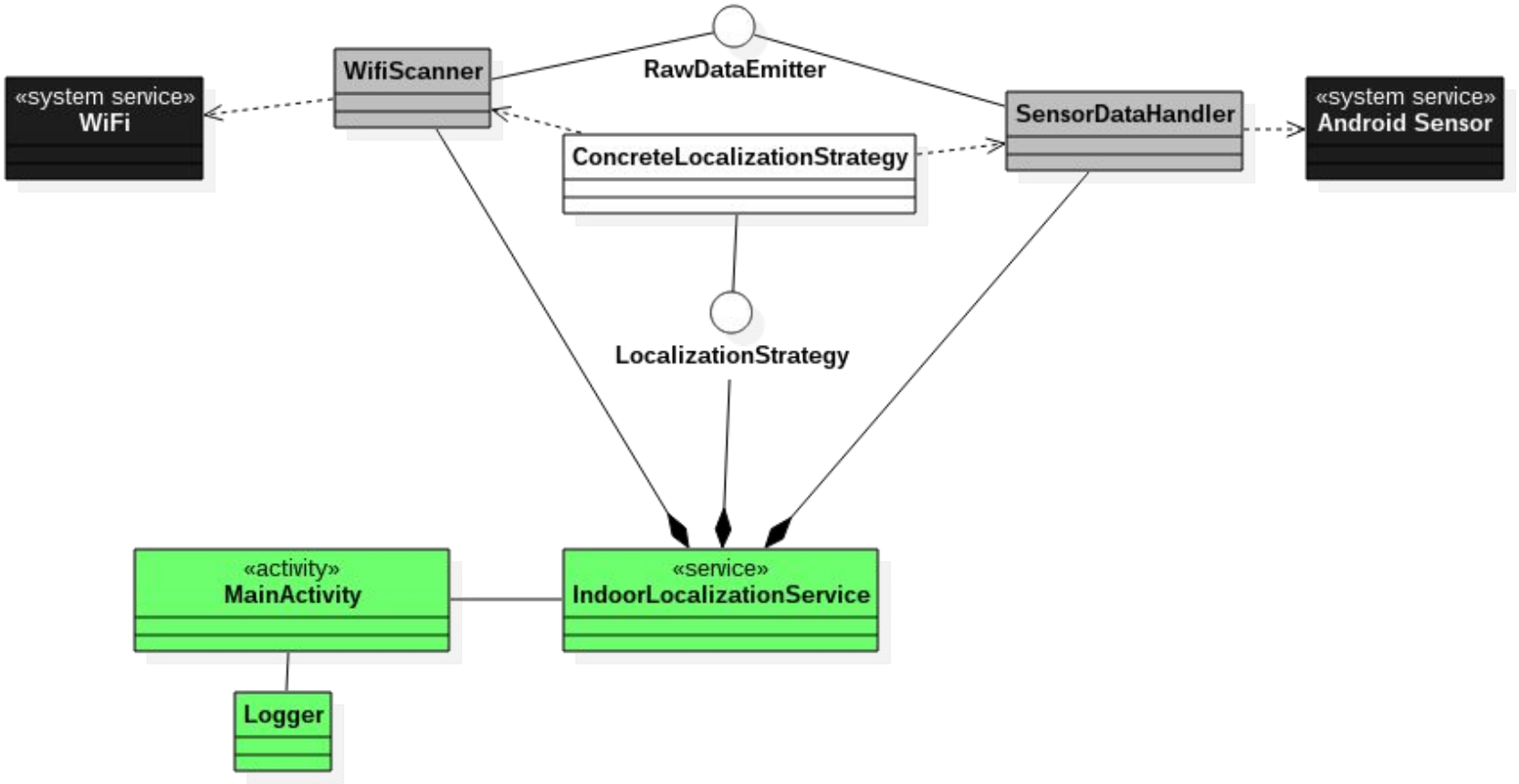
Application modules:
Java layer



Application modules: Java+Android layers



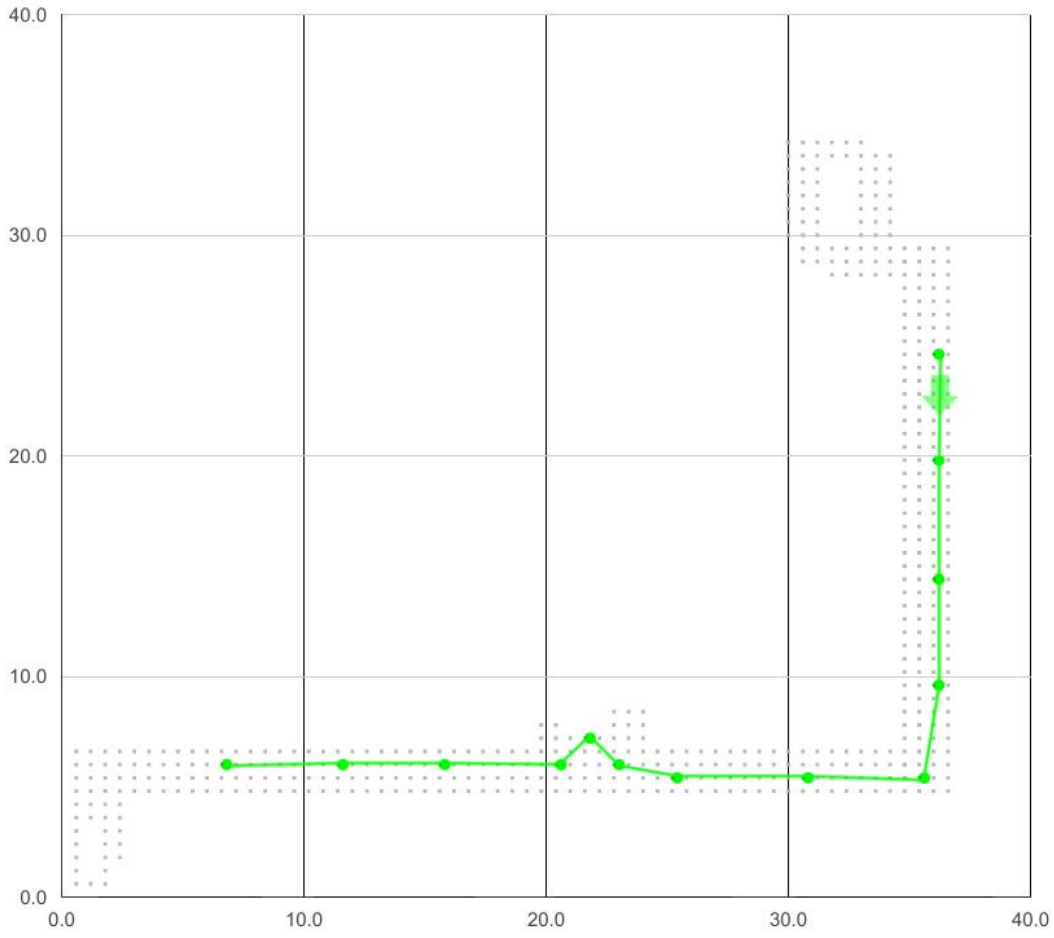
Application modules: Java+Android+Application layers



5,06m

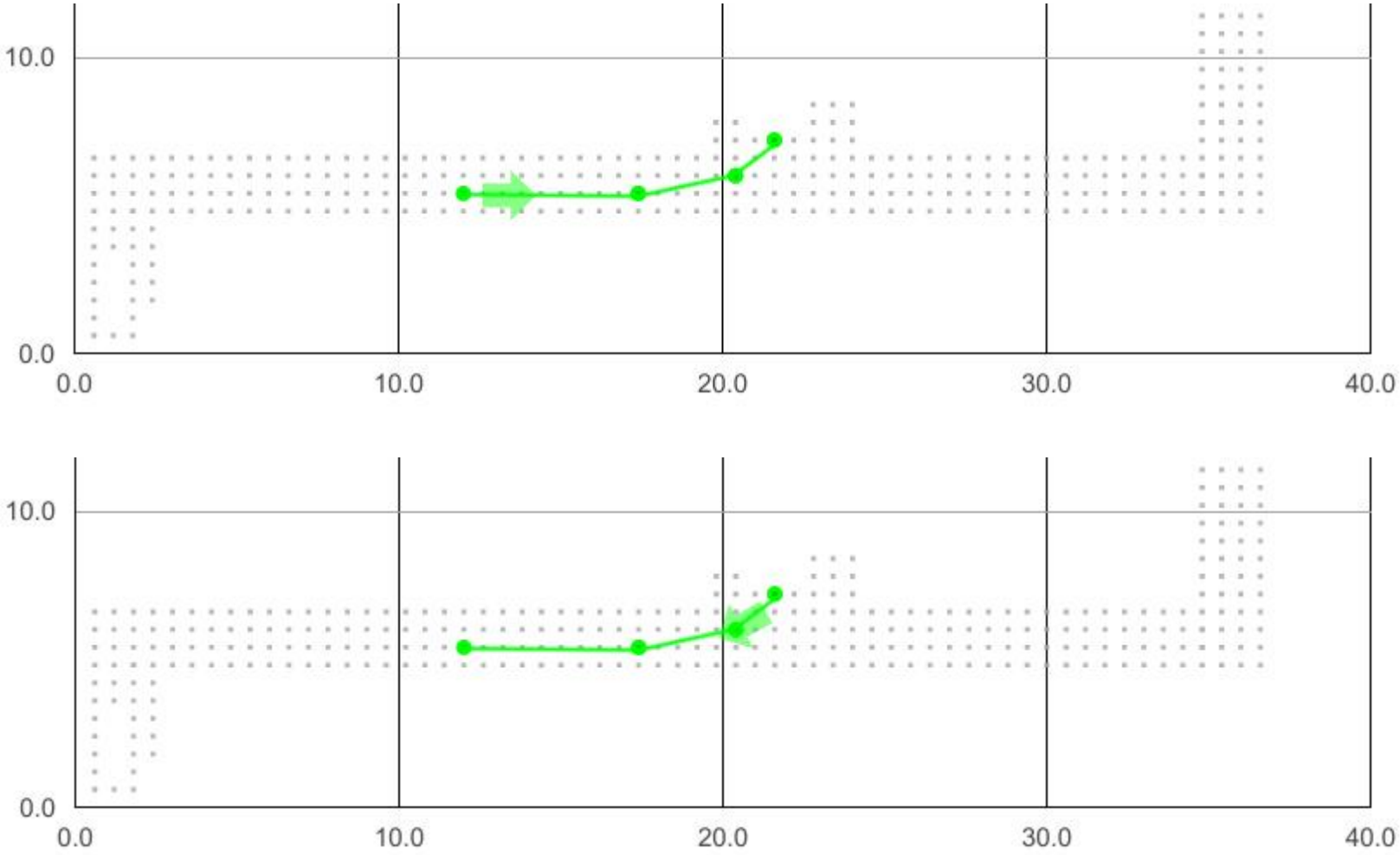
è il miglior terzo quartile.

Path 1



path credits: Paolo Barsocchi

Path 2



path credits: Paolo Barsocchi

Osservazioni

- PF ha alcuni comportamenti inaspettati
- KF hai dei bug
- Metodi di Fingerprint sono i migliori (mappe relativamente recenti)

La documentazione arriverà poco dopo il 28/4