Smart Waste Collection
Elaborato di progetto per il corso LSS

Martina Baiardi - Alessandro Marcantoni
Simone Romagnoli - Marta Spadoni

14/07/2022

Overview

Obiettivi del progetto

L'obiettivo principale del progetto è quello di ottimizzare la gestione della raccolta dei rifiuti utilizzando tecnologie innovative.

In particolare, si vuole:

- Dotare di sensori i cassonetti e i camioncini dei rifiuti, per poterne monitorare lo stato.
- Automatizzare la creazione delle missioni di raccolta, per renderle più efficienti.
- Migliorare la gestione dei rifiuti "straordinari", ossia quelli che richiedono una raccolta particolare.
- Monitorare lo stato di funzionamento dei servizi offerti dall'organizzazione.

Analisi del Problema

È stata condotta un'intervista con degli esperti di dominio che hanno consentito di chiarire quale sia l'attuale funzionamento del sistema. Questi hanno inoltre espresso i miglioramenti che desiderano vengano introdotti nella gestione della raccolta dei rifiuti.

Grazie alle risposte ottenute è stato possibile costruire un **Ubiquitous Language** condiviso tra il team di sviluppo e l'azienda.

Ubiquitous Language

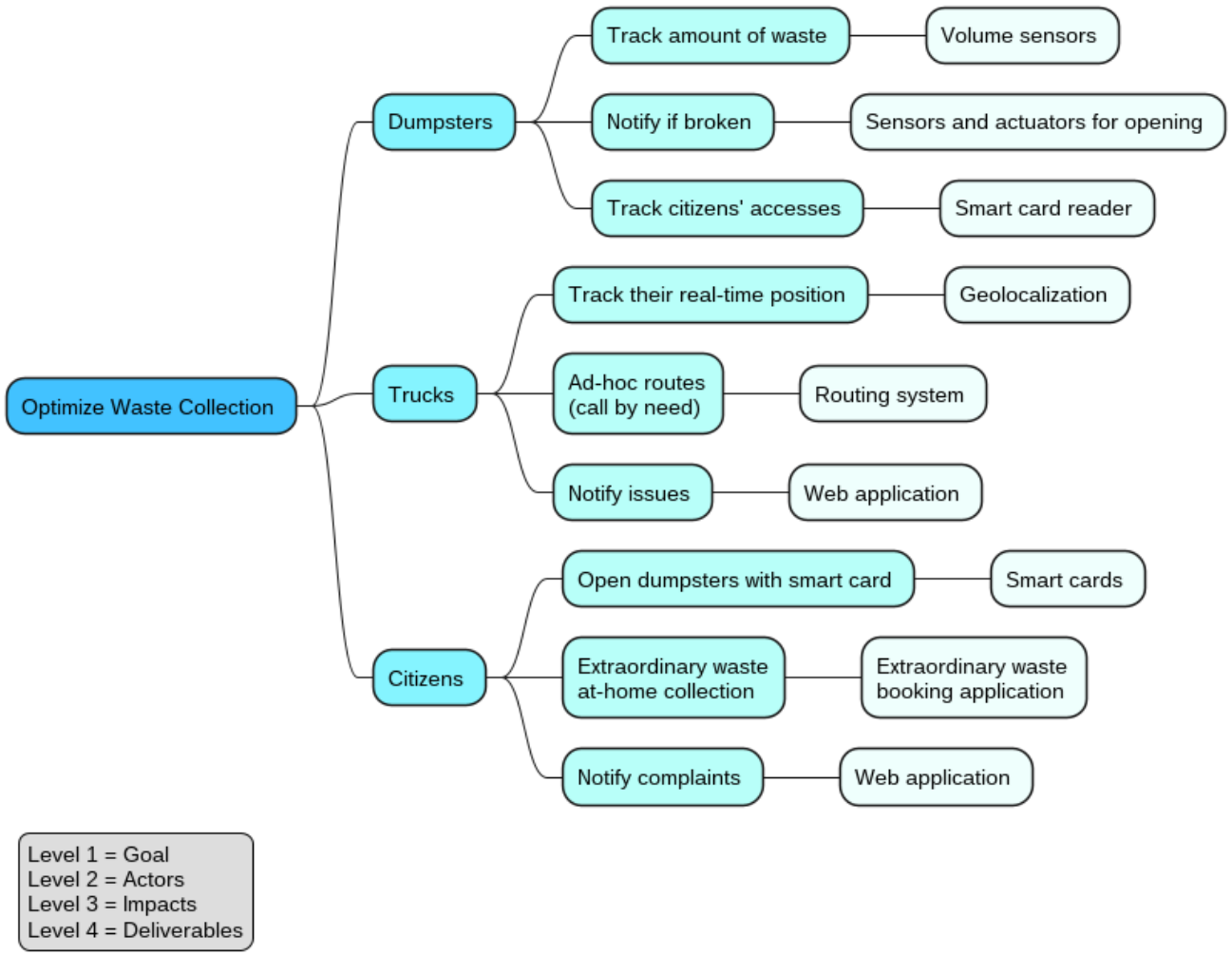
Grazie all'appronfondita analisi del dominio che è stata effettuata con i committenti, il team ha prodotto un **Ubiquituos Language**.

La costruzione di questo vocabolario ha consentito a tutti i membri del team di avere chiaro quali siano gli elementi che sono presenti all'interno del dominio e quali siano i loro ruoli, rimuovendo tutte le ambiguità.

L'Ubiquituos Language ha consentito agli sviluppatori di lavorare in modo indipendente e parallelo su componenti differenti del sistema.

Analisi dell'Impatto

Per fare chiarezza con l'organizzazione sull'impatto del progetto è stato realizzato un **Impact Mapping**.



Requisiti

Per definire i requisiti del sistema che verrà sviluppato, sono state utilizzate due tecniche:

- **User Stories**: Per delineare quali sono i requisiti richiesti dal punto di vista degli utilizzatori del sistema.
- **Use Cases**: Per una rappresentazione tramite *diagrammi dei casi d'uso* per evidenziare le interazioni tra i vari attori del sistema.

User Stories

Le **User Stories** sono state definite dal punto di vista di **Managers**, **Citizens** e **Truck Drivers**, seguendo il pattern:

```
As a ...  I want to ...  So that ...
```

Ad esempio:

> ***As a*** *Citizen* ***I want to*** *book an "at home" waste collection* ***so that*** *I don't have to go to the disposal point.*

Use Cases

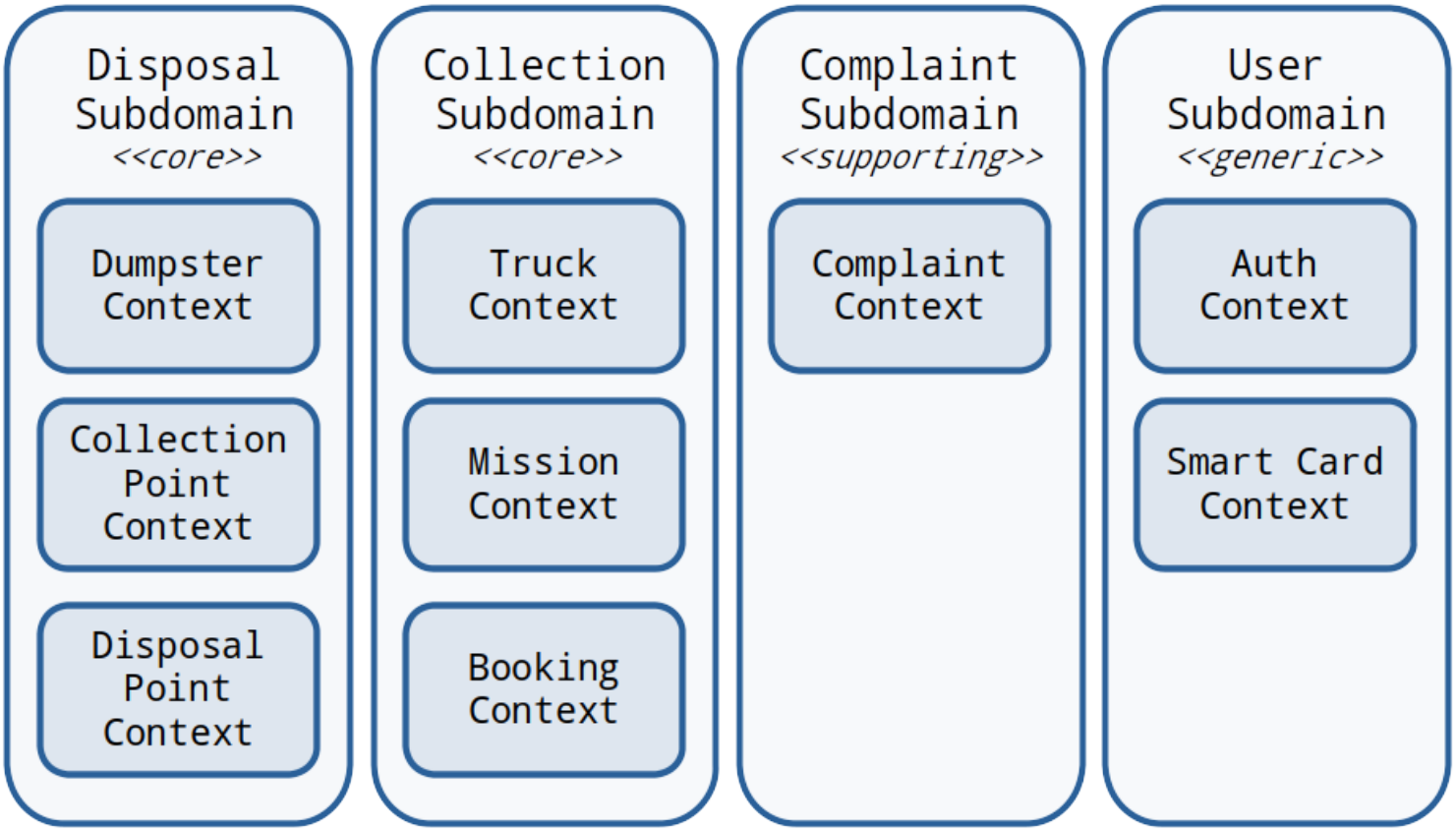
Dopo aver analizzato le funzionalità dal punto di vista degli attori del sistema, sono stati definiti dei diagrammi UML che ne descrivono i comportamenti.

In particolare sono stati evidenziati le seguenti ”macro-funzionalità”:

- Gestione della raccolta dei **rifiuti ordinari**
- Gestione della raccolta dei **rifiuti straordinari**
- **Dashboard**
- Gestione dei **reclami**

Bounded Contexts

Dopo aver effettuato la fase di analisi, il team ha delineato i **Subdomain** e i **Bounded Context** che compongono il dominio.



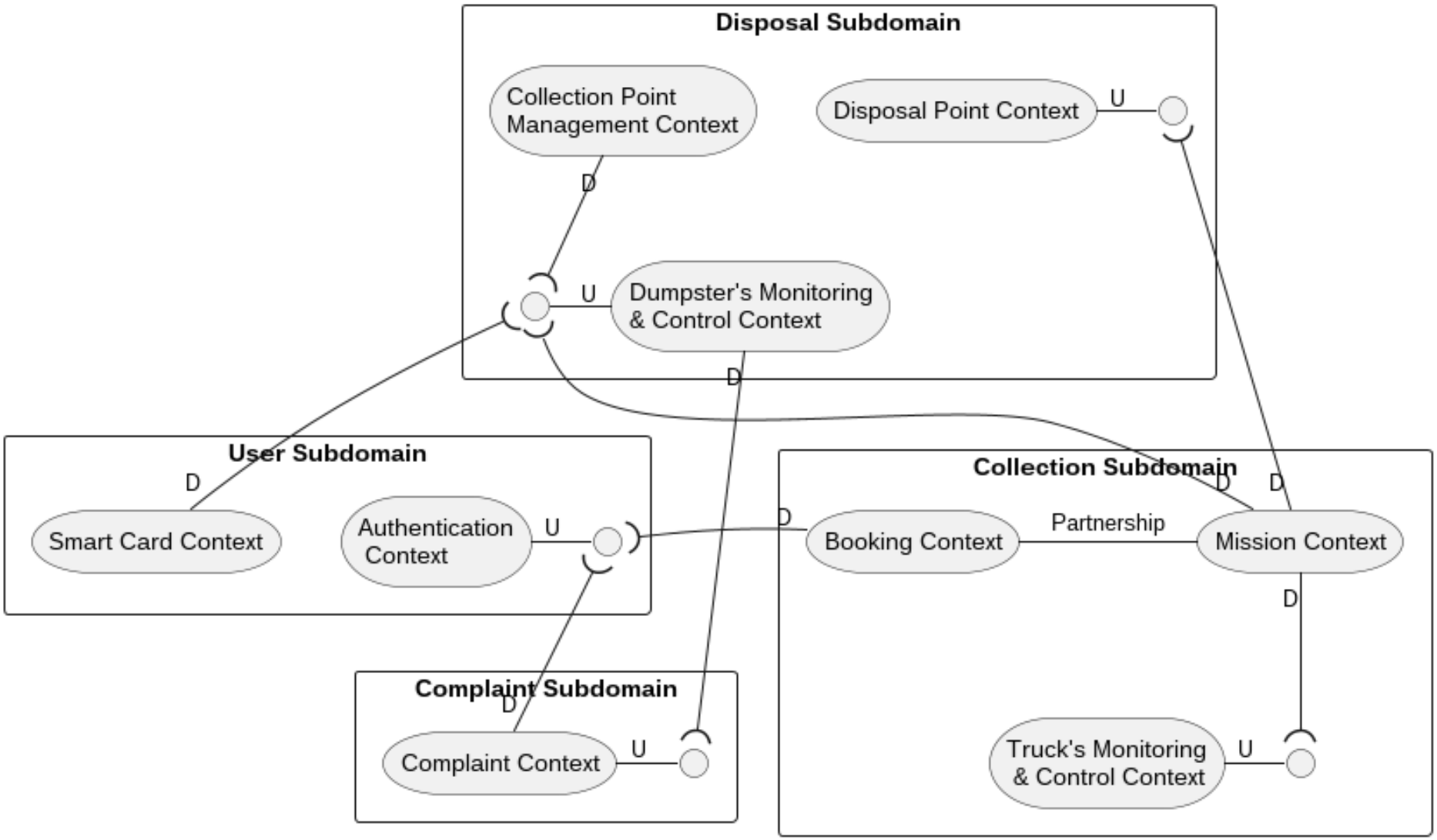
Context Mapping

Una volta individuati i **Subdomain** sono state analizzate le possibili relazioni che avvengono tra di essi.

In particolare, sono state individuate due tipologie di interazione:

- **Partnership**: dove i componenti si accordano sull'integrazione delle funzionalità
- **Upstream - Downstream (Conformist)**: dove il downstream si adatta alla specifica api definita dall'upstream

Context Mapping



Domain Models

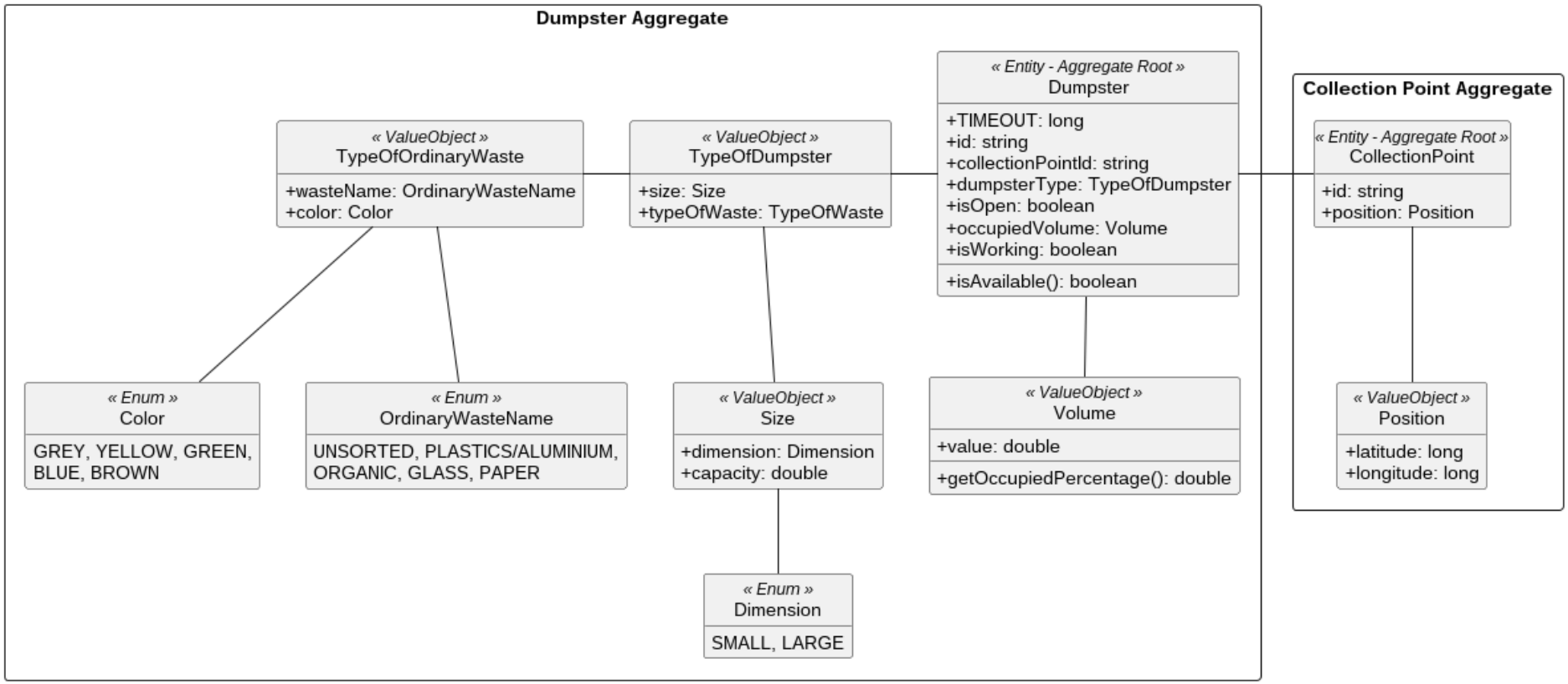
Sono stati modellati gli elementi *core* del dominio utilizzando i **Tactical Building Blocks**, definendo i principali **Aggregates**.

Gli aggregates modellati sono:

- **Dumpster Aggregate**
- **Collection Point Aggregate**
- **Truck Aggregate**
- **Mission Aggregate**

All'interno di ciascuno di essi sono stati specificati: *Entities*, *Value Objects*, *Domain Services* e *Domain Events*.

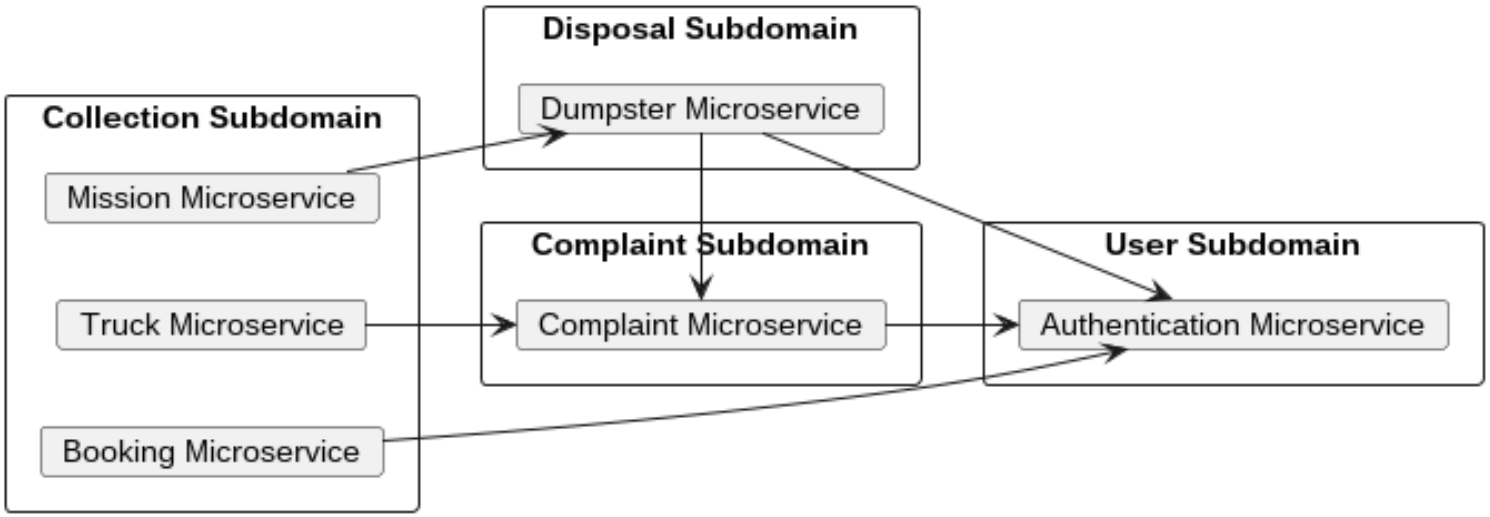
Domain Models



Architettura

È stata definita una architettura ad alto livello del sistema, a partire dai **Subdomain** precedentemente identificati.

Per ciascuno di essi, vengono individuati i *microservizi* necessari per il funzionamento del sistema, specificando anche le interazioni che avvengono tra essi.

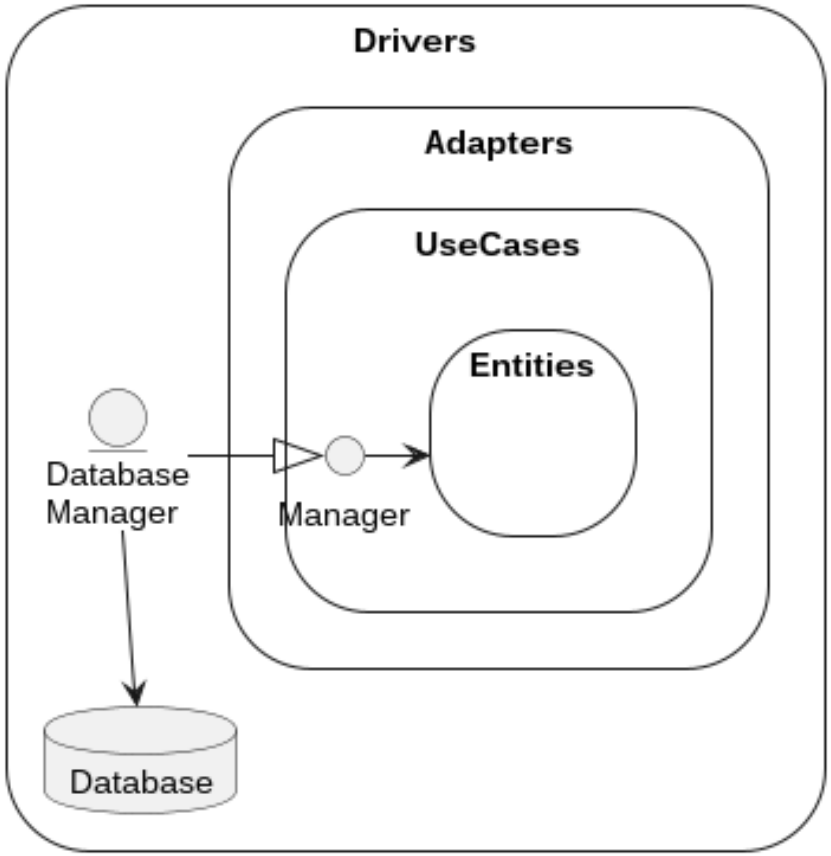


Architettura dei Microservizi

Ciascun microservizio è stato implementato ispirandosi alla **Clean Architecture**.

In particolare, sono state definiti i seguenti layer:

- **Entities**
- **Use Cases**
- **Adapters**
- **Drivers**



DVCS Strategy

Per gestire i repository sono state effettuate le seguenti scelte:

- Definizione di una **GitHub Organization** per contenere tutti i progetti.
- Utilizzo di **Git Flow Workflow** per avere una linea di sviluppo chiara e ben definita.
- Utilizzo dei **Conventional Commit** per poter generare in modo automatico i nuovi **tag** e le relative **release**
- Configurazione di un **Commit Linter** per assicurare che i commit siano correttamente formattati

Continuous Integration

All'interno dei repository dell'organizzazione sono stati configurati dei workflow che, tramite delle GitHub Actions, effettuano i seguenti controlli:

- **Code Quality Control**: tramite i plugin `ktlint` e `ESLint`.
- **Testing**: utilizzando i plugin `kotest` e `mocha`.
- **Reporting della coverage**: con i plugin `jacoco` e `istanbul`.

Inoltre è stato configurato il bot `renovate` all'interno dei repository per effettuare **Automatic Dependency Update**.

Continuous Delivery

Sono stati inoltre configurati dei workflow che consentono di effettuare:

- **Semantic Versioning and Releasing**: GitHub Action che consente di calcolare automaticamente il tag della release ed effettuarla tramite l'analisi dei messaggi di commit.
- **Containerization**: L'applicazione viene costruita utilizzando un Dockerfile e pubblicata all'interno dei GitHub Packages.