Gestione di un sistema di book-crossing: *The WalkingBooks*

Documentazione progettuale

Progetto del corso di Informatica IIIB
A.A. 2018/2019

Paganessi Andrea
Piffari Michele
Villa Stefano

February 6, 2020

Contents

List of Figures

List of Tables

Part I

Iterazione 0

1

Requisiti e specifiche

1.1 Requisiti utente

La startUp bergamasca Book Crossing UniBg desidera mettere a disposizione dei propri utenti un'applicazione Android per poter gestire la libera condivisione di libri all'interno di una vasta community di utenti.

I libri della rete di BookCrossing che l'azienda punta a gestire si possono trovare

- In un qualsiasi luogo (in stazione, su una panchina, in un locale...): funzionalità *"On The Go"*
- Nella zona di scambio ufficiale (*"OCZ UniBg"*: Official Crossing Zone UniBg)

Per il momento, l'unica OCZ gestita direttamente dalla startUp si trova all'interno dell'aula studio del campus di Ingegneria di Dalmine, la quale coincide anche con la sistemazione del server centrale che andrà a gestire i vari interscambi tra gli users.

La startUp richiede che, per usufruire dell'applicativo mobile, i clienti debbano registrarsi fornendo i propri dati quali:

- Nome
- Cognome
- Contatto di riferimento (opzionale)
  - Numero telefonico
  - Indirizzo mail
  - Facebook
  - ID Twitter
- Categorie di libri preferite
- Zona di residenza
- Raggio d'azione (inteso come il raggio, in km, entro cui l'utente è disposto a spostarsi per incontrare altri utenti con cui effettuare uno scambio di libri)

Una volta terminata la registrazione, l'utente può partecipare al programma di Book Crossing.

Secondo la politica del book sharing, per rendere disponibile alla comunità uno o più libri che non sono ancora presenti nel network stesso, serve identificarli univocamente potendone così tracciare la storia, ovvero ciò che concerne il percorso seguito dal libro, le recensioni lasciate dagli utenti etc.

Prima di procedere con l'identificazione univoca del libro, l'utilizzatore deve inserire i dati del testo (o dei testi) che intende condividere con il resto della community.

Questo inserimento può avvenire

- In maniera "automatica" tramite scansione del codice ISBN
- In modalità "manuale", nel caso in cui, per esempio, non sia presente il barcode

andando a fornire i seguenti dati:

- Titolo
- Autore
- Anno di pubblicazione/Edizione
- Categoria

A questo punto il sistema genererà un BCID di 10 caratteri, ovvero un *Book Crossing ID* univoco, il quale dovrà essere riportato sul testo dall'utente.

La vera e propria condivisione avviene nel momento in cui il volume viene rilasciato (azione che può avvenire in un secondo momento rispetto alla fase di identificazione), quando il sistema dovrà acquisire i seguenti dati:

- Luogo di rilascio (con estensione future per un'acquisizione automatica della posizione tramite GPS)
- Ora e data di rilascio

L'app inoltre consiglierà all'utente un luogo di rilascio in cui sia già presente almeno un libro, facilitando così la creazione di cassette virtuali, ovvero di luoghi in cui sono presenti più libri: l'idea è quella quindi di permettere al sistema di condivisione di creare, in maniera autonoma, dei punti "fissi" di consegna senza dover applicare interventi a livello infrastrutturale.

Successivamente il sistema dovrà notificare gli utenti, interessati al genere del libro rilasciato, della presenza di un nuovo testo appena rilasciato che potrebbe interessargli.

In qualsiasi momento è possibile effettuare le seguenti operezioni su ogni libro personalmente condiviso con la rete di sharing:

- Aggiunta di recensione
- Rating del libro

Quando viene trovato un libro (nel gergo definito come *"journal entry"*), il cliente che vuole prelevarlo, dopo aver effettuato il login nell'applicazione, deve inserire nell'apposito menù il BCID del libro che intende acquisire. Il sistema si occuperà poi di informare la community aggiornando lo status del libro raccolto, che diventerà *"underReading"*.

Per quanto concerne invece l'area riservata, ogni utente ha la possibilità di visualizzare informazioni in merito ai libri che:

- ha messo a disposizione della community (**relased**)
- ha ottenuto dalla community (**chased**)
- attualmente possiede

L'utente può effettuare la prenotazione di libri già inseriti nella liste *"chased"* e *"relased"* del proprio profilo.

Il sistema deve prevedere anche la possibilità di ricercare un specifico testo e visualizzare i contatti dei lettori del libro al fine di potersi scambiare opinioni e/o pareri in merito al libro stesso.

Tale funzionalità di ricerca permette anche la prenotazione del testo ricercato purché lo stesso sia nello stato *"under reading"*. Per soddisfare questa richiesta il sistema provvederà a consigliare, al lettore corrente del libro prenotato, zone di rilascio specifiche al fine di avvicinare tale libro al richiedente, tenendo presente anche la necessità di creare cassette virtuali (come specificato in precedenza).

2

Use cases

2.1 Analisi testuale dei casi d'uso

- ***UC1: Registration***
  - **Descrizione:** registrazione alla rete di Book Crossing.
  - **Attori coinvolti:** utente.
  - **Preconditions:**
    - smartphone dotato di connessione dati;
    - l'utente non è ancora registrato al programma di Book Crossing.
  - **Postconditions:** l'utente è registrato al programma di Book Crossing.
  - **Processo:**
    1. l'utente seleziona "Registrati" nella schermata inziale dell'applicazione;
    2. l'applicazione mostra un form in cui l'utente può inserire:
       a. Nome
       b. Cognome
       c. Data di nascita
       d. Username
       e. Password
       f. Contatto di riferimento
       g. Categorie di libri preferite
       h. Zona di residenza
       i. Raggio d'azione rispetto alla zona di residenza
    3. l'utente inserisce i dati richiesti nel form presentato;
    4. l'utente attende la visualizzazione della conferma di avvenuta registrazione;
    5. l'utente viene reindirizzato alla pagina principale dell'applicazione.
  - **Alternative**
    - **Dati non validi:** se l'utente inserisce dei dati non validi e/o mancanti, l'applicazione mostra un messaggio d'errore permettendo all'utente di modificare i dati non validi.
  - **Estensioni**
- ***UC2: Login***
  - **Descrizione:** accesso alla rete di Book Crossing.
  - **Attori coinvolti:** utente.
  - **Preconditions:**
    - smartphone dotato di connessione dati;

    - l'utente è già registrato al programma di Book Crossing.
  - **Postconditions:** l'utente è loggato nella rete di Book Crossing.
  - **Processo:**
    1. l'utente seleziona "Login" nella schermata iniziale dell'applicazione;
    2. l'applicazione mostra una schermata in cui l'utente inserisce username e password;
    3. l'utente inserisce i dati richiesti nella view presentata;
    4. l'utente attende la verifica della correttezza dei dati inseriti;
    5. l'utente viene reindirizzato alla pagina principale dell'applicazione.
  - **Alternative**
    - **Username e/o password non corretti:** se l'utente inserisce username e/o password non validi, l'applicazione mostra un messaggio d'errore permettendo all'utente di modificare i dati.
  - **Estensioni**

- ***UC3: Logout***
  - **Descrizione:** disconnessione profilo personale.
  - **Attori coinvolti:** utente.
  - **Preconditions:**
    - smartphone dotato di connessione dati;
    - l'utente è già registrato al programma di Book Crossing;
    - l'utente è loggato.
  - **Postconditions:** l'utente non è più loggato nella rete di Book Crossing.
  - **Processo:**
    1. l'utente seleziona "Profilo personale" nella schermata principale dell'applicazione;
    2. l'applicazione mostra una schermata in cui l'utente può visualizzare tutte le proprie informazioni;
    3. l'utente preme il bottone "Logout";
    4. l'utente viene reindirizzato alla pagina di login.

- ***UC4: Book pick-up***
  - **Descrizione:** raccolta di un libro "On The Go"[1] o in una OCZ.[2]
  - **Attori coinvolti:** utente.
  - **Preconditions:**
    - smartphone dotato di connessione dati;
    - l'utente ha effettuato l'accesso alla rete di Book Crossing;
    - il libro è stato siglato con il codice BCID.
  - **Postconditions:** Il libro viene associato all'utente.
  - **Processo:**
    1. l'utente seleziona "Raccogli libro" nel menu principale dell'applicazione;
    2. l'applicazione mostra un form in cui l'utente può inserire il BCID del libro;
    3. l'utente inserisce il codice BCID riportato nel libro;
    4. l'utente attende la visualizzazione di una scheda riepilogativa relativa al libro appena aggiunto;

[1] In questo caso il libro condiviso dalla community viene raccolto dall'utente in una qualsiasi zona (come per esempio la stazione, il parco, sale di attesa etc...).

[2] *Official Crossing Zone*: zone riconosciute e fisse in cui la community può liberamente scambiarsi i libri.

5. l'utente verifica la corrispondenza delle informazioni mostrate;
6. l'utente conferma la raccolta.

– **Alternative**

* **BCID inesistente:** se l'utente inserisce un BCID non esistente, l'applicazione mostra un messaggio d'errore permettendo all'utente di modificare il BCID.
* **BCID associato ad un altro utente:** se l'utente inserisce un BCID già in possesso di un'altro utente, l'applicazione mostra un messaggio d'errore permettendo all'utente di modificare il BCID.
* **BCID non corrispondente:** se l'utente, al punto (4), verifica che il libro reale non corrisponde alle informazioni mostrate dall'applicazione, può annullare l'operazione di raccolta.

– **Estensioni**

- ***UC5: Book registration***

– **Descrizione:** registrazione di un libro alla rete di Book Crossing (*journal entry*).

– **Generalizzazione di:**

* aggiunta manuale dei dati del libro (UC8);
* scansione ISBN (UC9).

– **Include:** scrittura BCID (UC10).

– **Attori coinvolti:** utente.

– **Preconditions:**

* smartphone dotato di connessione dati;
* l'utente ha effettuato l'accesso alla rete di Book Crossing;
* il libro non è ancora stato siglato con il codice BCID.

– **Postconditions:** Il libro viene registrato alla rete di Book Crossing.

– **Processo:**

1. l'utente seleziona "Registra un nuovo libro" nel menu principale dell'applicazione;
2. l'applicazione tenta di iniziare la scansione ISBN mostrando all'utente la fotocamera;
3. l'utente inquadra il codice ISBN del libro per il tempo sufficiente al riconoscimento del codice stesso;
4. l'applicazione mostra all'utente la schermata contente tutti le informazioni del libro;
5. l'utente preme il pulsante "Conferma registrazione", dopo aver verficato rapidamente la coerenza dei dati.

– **Alternative**

* **Aggiunta manuale dei dati:** se, dalla schermata di scansione, l'utente decide di inserire manualmente i dati del libro da registrate, l'applicazione mostra una schermata dove aggiungere manualmente i dati del libro.
* **Scansione fallita:** se la scansione fallisce, l'applicazione riapre la fotocamenra permettendo all'utente di ripetere l'operazione.

– **Estensioni**

- ***UC6: Book research***

– **Descrizione:** ricerca di un libro all'interno della rete di Book Crossing.

– **Attori coinvolti:** utente.

– **Preconditions:**

* smartphone dotato di connessione dati;

    - l'utente ha effettuato l'accesso alla rete di Book Crossing;
    - il libro è presente nella rete di Book Crossing.
  - **Postconditions:** il libro ricercato viene mostrato.
  - **Processo:**
    1. l'utente seleziona “Ricerca libro” nel menu principale dell'applicazione;
    2. l'applicazione mostra all'utente un form da completare in cui inserire i parametri della ricerca;
    3. l'utente inserisce i parametri a cui è interessato;
    4. l'utente preme il pulsante "Cerca" ed attende il completamento;
    5. l'utente visualizza la lista di tutti i libri nella rete, che soddisfano la ricerca.
    6. il sistema verifica la presenza del libro cercato;
    7. in caso di esito positivo, l'applicazione mostra una scheda riassuntiva del libro;
    8. in caso di esito negativo, l'applicazione mostrerà un messaggio di errore.
  - **Alternative**
    - **Parametri non validi:** se l'utente inserisce dei parametri non validi, l'applicazione mostra un messaggio d'errore permettendo all'utente di modificarli.
    - **Ricerca senza risultati:** se la ricerca non va a buon fine, l'applicazione mostra un messaggio all'utente, comunicando che nessun libro presente nella rete soddisfa i parametri di ricerca inseriti.
  - **Estensioni**
    - L'utente può selezionare uno dei libri mostrati dall'applicazione e visualizzare le sue informazioni.

- ***UC7: Info visualization***
  - **Descrizione:** Visualizzazione informazioni
  - **Generalizzazione:** visualizzazione informazioni libri chased (UC13), visualizzazione informazioni libri relase (UC14) e visualizzazione informazioni libri in possesso (UC15).
  - **Attori coinvolti:** utente.
  - **Preconditions:**
    - smartphone dotato di connessione dati;
    - l'utente ha effettuato l'accesso alla rete di Book Crossing.
  - **Postconditions:** L'applicazione mostra le informazioni desiderate
  - **Processo:**
    1. l'utente seleziona la voce “I miei libri” nel menu principale dell'applicazione;
    2. l'applicazione mostra categorie di informazioni visualizzabili;
    3. l'utente seleziona la categoria che vuole visualizzare;
    4. l'applicazione mostra l'elenco dei libri della categoria selezionata.
  - **Alternative**
    - **Nessun libro in elenco:** se nessun libro è presente nello storico, l'applicazione mostra un messaggio all'utente, comunicando che non è stata ancora effettuata nessuna operazione nella comunità.
  - **Estensioni**

- ***UC8: Personal area visualization***
  - **Descrizione:** Visualizzazione profilo personale utente
  - **Attori coinvolti:** utente.

  - **Preconditions:**
    * smartphone dotato di connessione dati;
    * l'utente ha effettuato l'accesso alla rete di Book Crossing;
  - **Postconditions:** l'applicazione mostra il profilo dell'utente.
  - **Processo:**
    1. l'utente seleziona la voce "Il mio profilo" nel menu principale dell'applicazione;
    2. l'applicazione mostra l'anagrafica, i contatti e le attività svolte dall'utente;
  - **Estensioni:**

- ***UC9: Manual addition of book's data***
  - **Descrizione:** Inserimento manuale di un libro nella rete di Book Crossing
  - **Attori coinvolti:** utente.
  - **Preconditions:**
    * smartphone dotato di connessione dati;
    * l'utente ha effettuato l'accesso alla rete di Book Crossing;
    * il libro non è stato ancora siglato con il codice BCID.
  - **Postcondition:** Viene generato il codice BCID e il libro viene aggiunto alla rete di Book Crossing
  - **Processo:**
    1. facendo riferimento al passo 1 e 2 del UC4, l'utente preme il pulsante "Aggiunta manuale";
    2. l'applicazione mostra un form da compilare con i dati del libro;
    3. l'utente inserisce i dati del libro richiesti e conferma l'operazione;
    4. l'applicazione mostra il codice BCID da trascrivere sul libro;
    5. il sistema aggiunge il libro alla rete di Book Crossing.
  - **Estensioni**

- ***UC10: ISBN scan***
  - **Descrizione:** scansione del codice ISBN tramite fotocamera per ottenere le informazioni in merito al libro da registrare.
  - **Attori coinvolti:** utente.
  - Preconditions:
    * smartphone dotato di connessione dati;
    * l'utente ha effettuato l'accesso alla rete di Book Crossing;
    * il libro possiede il codice ISBN.
  - **Postconditions:** il libro è in possesso dell'utente e non più condiviso con la community.
  - **Processo:**
    1. l'utente seleziona "Registra un nuovo libro" nel menu principale dell'applicazione;;
    2. Viene aperta la fotocamera all'interno dell'applicazione;
    3. l'utente inquadra il codice ISBN finchè il sistema non rileva il barcode.
  - **Alternative**
    * **ISBN non riconsciuto:** il sistema non è in grado di riconoscere l'ISBN inquadrato. Si chiuderà la fotocamera e l'utente verrà reindirizzato alla pagina di inserimento manuale del libro (UC9).
  - **Estensioni**

- ***UC11: Instruction to write BCID code***[3]
  - **Descrizione:** scrittura del codice identificativo sul libro condiviso.
  - **Attori coinvolti:** utente
  - **Preconditions:**
    * smartphone dotato di connessione dati;
    * l'utente ha effettuato l'accesso alla rete di Book Crossing;
  - **Postconditions:** il libro è univocamente riconosciuto del sistema tramite il BCID.
  - **Processo:** l'utente copia il codice BCID sul libro, seguendo le istruzioni mostrate dall'applicazione.
  - **Estensioni**
- ***UC12: View of users contacts***
  - **Descrizione:** l'utente ottiene i contatti che un altro utilizzatore ha deciso di condividere con la community.
  - **Attori coinvolti:** utente.
  - **Preconditions:**
    * smartphone dotato di connessione dati;
    * l'utente ha effettuato l'accesso alla rete di Book Crossing.
  - **Postconditions:** l'applicazione visualizza i contatti a cui l'utente può e/o vuole essere contattato.
  - **Processo:**
    1. l'utente selezione "Ricerca" dal menu principale dell'applicazione;
    2. l'utente seleziona il libro a cui è interessato;
    3. il sistema mostra tutte le informazioni relative al libro (tra cui anche la lista di tutti i lettori che sono stati in possesso del libro in questione);
    4. l'utlizzatore seleziona l'utente che desidera contattare;
    5. il sistema mostra tutte le informazioni di contatto che il cliente terzo ha deciso di condividere con la community.
  - **Alternative:**
    * **Nessun libro trovato:** l'applicazione notifica l'utilizzatore del fatto che la ricerca non sia andata a buon fine.
    * **Utente senza alcun contatto condiviso:** il sistema filtra la visualizzazione della lista degli utenti, visualizzando solo coloro che hanno inserito, durante la fase di registrazione, almeno un contatto o che desiderano essere contattati.
  - **Estensioni**
- ***UC13: Book reservation***
  - **Descrizione:** l'utilizzatore prenota un determinato libro in possesso di un altro lettore.
  - **Attori coinvolti:**
    * utente richiedente (*claimant user*);
    * utente attualmente in possesso del libro richiesto (*owner user*).
  - **Preconditions:**
    * smartphone dotato di connessione dati;
    * l'utente ha effettuato l'accesso alla rete di Book Crossing;
    * il libro che si vuole prenotare deve essere registrato alla rete;

[3] *Book Crossing IDentifier*

    - il libro richiesto deve essere già in possesso di un altro utente.
  - **Postconditions:**
    - Se i due utenti hanno un punto di incontro in comune, si accordano sul luogo di scambio.
    - Se invece non hanno un punto di incontro comune, il libro passerà tra gli utenti che si trovano tra *claimant user* e *owner user*.
  - **Processo:**
    1. l'utente seleziona "Ricerca libro" nel menu principale dell'applicazione;
    2. l'applicazione mostra all'utente un form da completare in cui inserire i parametri della ricerca;
    3. l'utente inserisce i parametri a cui è interessato;
    4. l'utente preme il pulsante "Cerca" ed attende il completamento;
    5. l'utente visualizza la lista di tutti i libri nella rete, che soddisfano la ricerca.
    6. il sistema verifica la presenza del libro cercato;
    7. l'utente va a selezionare il libro all'interno della lista proposta dal sistema;
    8. l'applicazione mostra un riepilogo sulle informazioni del libro, unitamente alla possibilità di prenotare;
    9. l'utente preme il pulsante "Prenota";
    10. l'applicazione mostra una pagina di conferma dell'avvenuta prenotazione.
  - **Alternative**
    - **Libro non prenotabile:** il libro selezionato non è prenotabile poichè non in possesso di un altro utente.
  - **Estensioni**

- ***UC14: Chased books informations***
  - **Descrizione:** visualizzazione storico dei libri "raccolti" dall'utente.
  - **Attori coinvolti:** utente.
  - **Preconditions:**
    - smartphone dotato di connessione dati;
    - l'utente ha effettuato l'accesso alla rete di Book Crossing.
  - **Postconditions:** mostrata sulla grafica la lista dei libri raccolti, con relativa data e luogo di "chasing".
  - **Processo:**
    1. l'utente seleziona "Il mio profilo" dal menù principale dell'applicazione;
    2. l'applicazione mostra le informazioni dell'utente e i diversi stati in cui si possono trovare i suoi libri;
    3. l'utente sceglie lo stato "Libri chased";
    4. l'applicazione mostra un riepilogo di tutti i testi ottenuti dalla community di sharing.
  - **Alternative:**
    - **Nessun libro chased:** l'applicazione mostra un messaggio all'utente, comunicando che nessun libro è stato ancora raccolto da lui dalla community.
  - **Estensioni**

- ***UC15: Released books informations***
  - **Descrizione:** visualizzazione storico dei propri libri condivisi con la rete.
  - **Attori coinvolti:** utente.
  - **Preconditions:**

    - ∗ smartphone dotato di connessione dati;
    - ∗ l'utente ha effettuato l'accesso alla rete di Book Crossing.
  - **Postconditions:** mostrata sulla grafica la lista dei libri inseriti nel programma di sharing, con relativa data e luogo di "relase".
  - **Processo:**
    1. l'utente seleziona "Il mio profilo" dal menù principale dell'applicazione;
    2. l'applicazione mostra le informazioni dell'utente e i diversi stati in cui si possono trovare i suoi libri;
    3. l'utente sceglie lo stato "Libri relased";
    4. l'applicazione mostra un riepilogo di tutti i libri rilasciati alla community di sharing.
  - **Alternative:**
    - ∗ **Nessun libro released:** l'applicazione mostra un messaggio all'utente, comunicando che nessun libro è stato ancora rilasciato da lui nella rete di Book Crossing.
  - **Estensioni**

- ***UC16: "Under reading" books informations***
  - **Descrizione:** visualizzazione dei propri libri attualmente "under reading".
  - **Attori coinvolti:** utente.
  - **Preconditions:**
    - ∗ smartphone dotato di connessione dati;
    - ∗ l'utente ha effettuato l'accesso alla rete di Book Crossing.
  - **Postconditions:** l'applicazione mostra la lista dei libri attualmente in possesso dell'utente
  - **Processo:**
    1. l'utente seleziona "Il mio profilo" dal menù principale dell'applicazione;
    2. l'applicazione mostra le informazioni dell'utente e i diversi stati in cui si possono trovare i suoi libri;
    3. l'utente sceglie lo stato "Libri in possesso";
    4. l'applicazione mostra un riepilogo di tutti i libri attualmente in possesso.
  - **Alternative:**
    - ∗ **Nessun libro in lettura:** l'applicazione mostra un messaggio all'utente, comunicando che nessun libro è in suo possesso al momento.
  - **Estensioni**

- ***UC17: Book relased***
  - **Descrizione:** l'utente libera un libro.
  - **Attori coinvolti:** utente.
  - **Preconditions:**
    - ∗ smartphone dotato di connessione dati;
    - ∗ l'utente ha effettuato l'accesso alla rete di Book Crossing;
    - ∗ libro rilasciato già registrato al sistema di Book Crossing.
  - **Postconditions:** il libro passa dallo stato "under reading" a quello "Available".
  - **Processo:**
    1. l'utente seleziona "Il mio profilo" dal menù principale dell'applicazione;
    2. l'applicazione mostra le informazioni dell'utente e i diversi stati in cui si possono trovare i suoi libri;
    3. l'utente sceglie lo stato "Libri in possesso";

4. l'applicazione mostra un riepilogo di tutti i libri attualmente in possesso;
5. l'utente preme sul testo che intende rilasciare;
6. l'utente conferma il rilascio.
7. l'applicazione mostra una conferma di avvenuto rilascio del testo selezionato.

– **Alternative:**
  * **Nessun libro in lettura:** l'applicazione mostra un messaggio all'utente, comunicando che nessun libro è in suo possesso al momento.
  * **Segnale GPS non trovato:** l'applicazione avvisa l'utente di attivare il GPS del dispositivo.

– **Estensioni**

2.2 Use Case Diagram

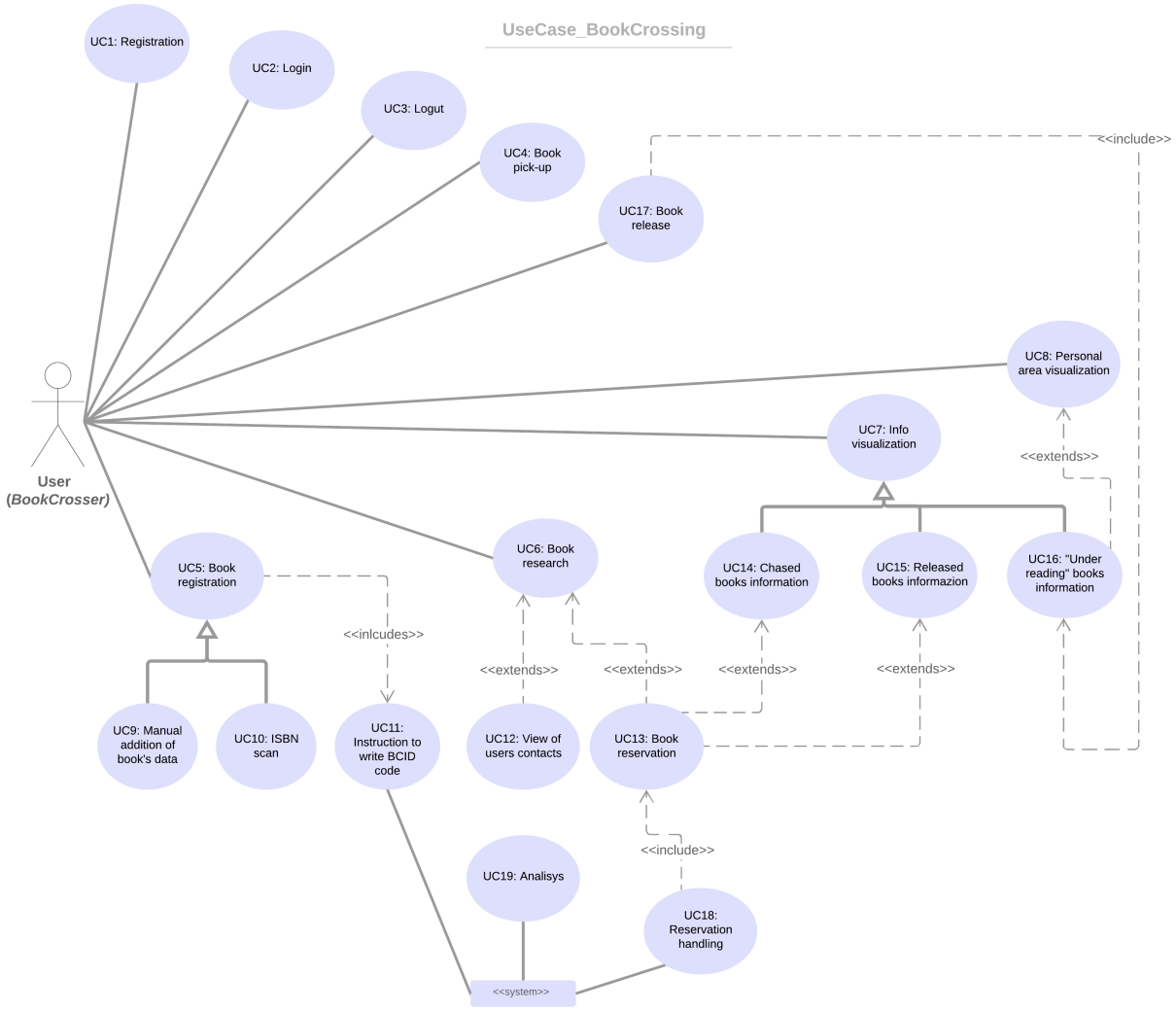

Figure 2.1: Use cases diagram

2.3 Funzionalità richieste

Table 2.1: Panoramica dei requisiti funzionali progettuali

Nome requisito	ID requisito	Tipologia	Priorità	Requisiti padre	Requisiti figli
Raccolta libro	UR1	funzionale	alta		UR2
Login utente	UR2	funzionale	alta	UR1	UR3, UR4
Registrazione utente	UR3	funzionale	alta	UR2	
Aggiunta libro	UR4	funzionale	alta	UR2	UR7, UR9
Ricerca libro	UR5	funzionale	media	UR2	UR10
Prenotazione libro	UR6	funzionale	bassa	UR2	
Visualizzazione info libri chased	UR7	funzionale	bassa	UR2	
Visualizzazione info libri released	UR8	funzionale	bassa	UR2, UR4	
Rilascio libro	UR9	funzionale	alta	UR2, UR4	
Visualizzazione contatti utenti	UR10	funzionale	bassa	UR2, UR5	
Visualizzazione profilo personale	UR11	funzionale	media	UR2	

Table 2.2: Descrizione requisiti non funzionali progettuali

Nome requisito	ID requisito	Descrizione
Controllo geolocalizzazione	UR12	Requisito non funzionalità che permette di verificare che la posizione GPS salvata del libro corrisponda, con margine d'accettazione, alla posizione in cui si trova l'utente nel momento in cui vuole raccogliere un libro trovato "On The Go".

2.4 Stati del libro

La starup si impone l'obbiettivo di andare a gestire lo scambio di libri all'interno della rete di book-crossing: come visto all'interno dei diversi use-cases, ogni libro nel corso della propria vita all'interno della community, passa di mano in mano attraversando diverse zone. A questo movimento fisico corrisponde anche un continuo cambio di stato da parte del libro stesso: possiamo riassumere con una *"Finite State Machine"* il percorso che un generico libro segue durante la sua vita.

Riassumendo gli stati di un libro, possono essere:

- Out of the network
- Available
- Under reading
- Released
- Reserved

- Traveling

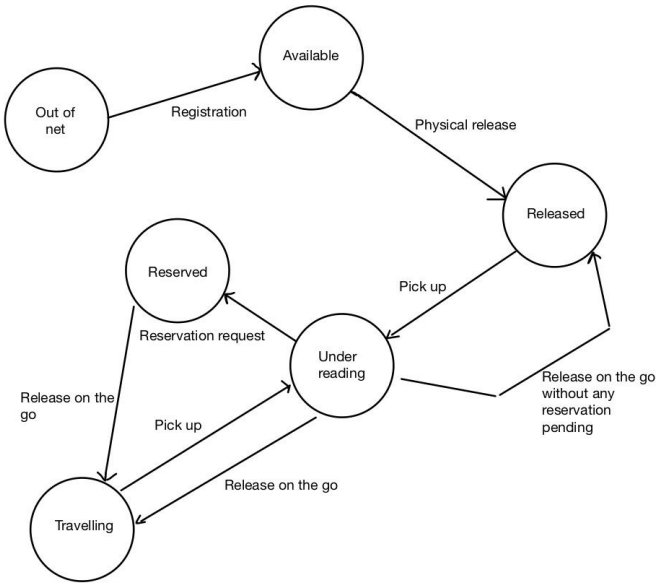

Figure 2.2: Stati del libro all'interno della community

2.5 Tool chain

Per la realizzazione del software presentato in questo report sono stati utilizzati i seguenti tool:

- **Modellazione**
  - **Use case diagram**: realizzato con il tool online *Lucid Chart* (https://www.lucidchart.com);
  - **Database architecture**: realizzato con *Vertabelo* (https://www.vertabelo.com/);
  - **Class diagram - deployment diagram - logical view**: EDrawMax (https://www.edrawsoft.com/en/max/) e Visual Paradigm (https://www.visual-paradigm.com/)
- **Implementazione software**
  - **Eclipse**: per quanto riguarda l'implementazione del codice lato server.
  - **Android Studio**: ambiente di sviluppo utilizzato per lo sviluppo della parte mobile.
- **Analisi del software**
  - **Espresso - JUnit** per l'analisi dinamica del codice
  - **CodeCover - SpotBugs** per l'analisi statica
- **Tool vari**
  - **Versioning**: repository Github gestito tramite interfaccia grafica;
  - **Documentazione**: LaTeXtramite interfaccia grafica TeXstudio.

3

Architettura

3.1 Deployment diagram

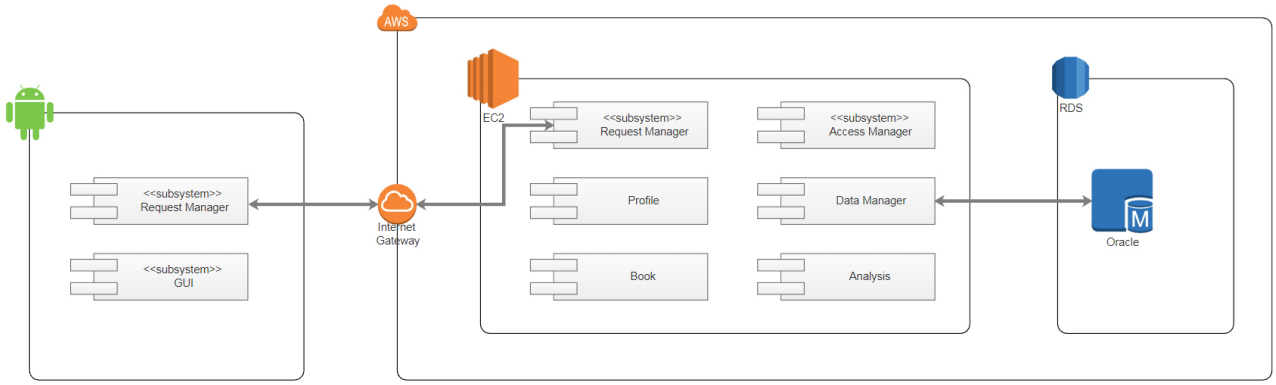

Figure 3.1: Deployment Diagram

3.2 Architecture Envisioning

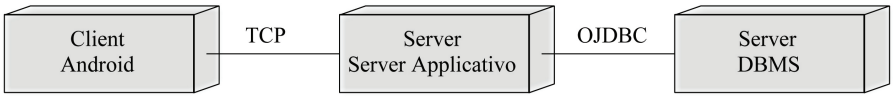

Figure 3.2: Architecture Envisioning

In figura 3.1 e 3.2 sono mostrati il Deployment Diagram e l'Architecture Envisioning del sistema progettato per lo sviluppo dell'applicazione di Book Crossing.

Si può osservare che si tratta di un'architettura ***Three Tiers***:

1. A sinistra si individua il *Client*, ovvero il dispositivo Android con il quale l'utente può interfacciarsi direttamente. Al suo interno si può osservare la presenza di un componente relativo all'interfaccia grafica e uno relativo alla gestione delle richieste per invio e ricezione di dati con il server;

2. Nella parte centrale individuiamo gli altri due layer dell'architettura: server EC2 e Database Relazionale RDS. Il fatto di utilizzare Amazon Web Services (AWS) consente di avere questi due elementi integrati in un unico strato.

Per la comunicazione tra smartphone e EC2 utilizziamo un connettore basato su una socket TCP, mentre il server applicativo (EC2) si connette al DBMS (RDS Oracle) tramite il connettore OJDBC (Oracle Java DataBase Connectivity).

3.3 Database architecture

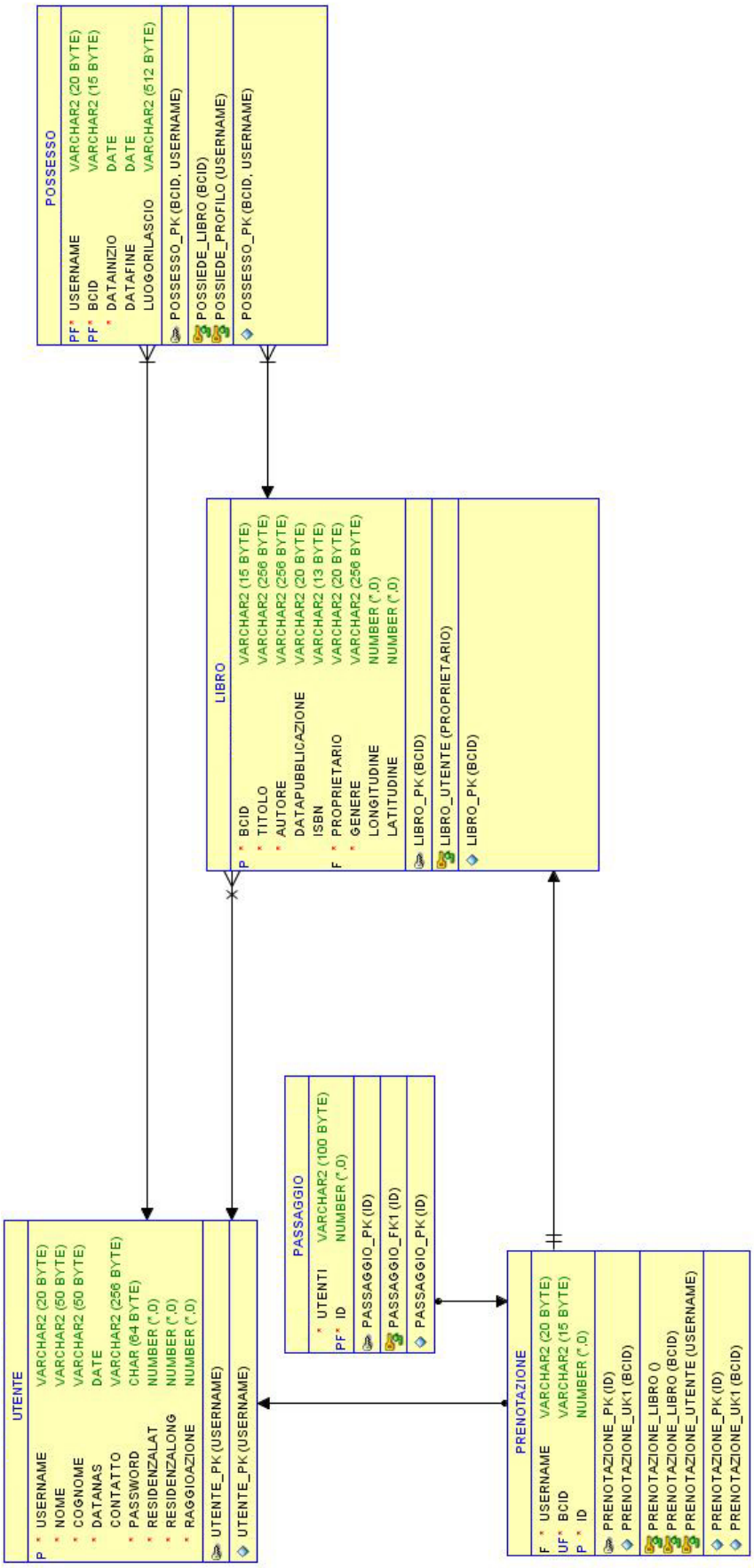

Figure 3.3: Modello del database

La figura 3.3 rappresenta il modello logico del database, le cui entità sono:

- **Utente** : contiene tutti i dati relativi all'utenza registata, ogni utente è identificato da uno *username* univoco;
- **Libro** : contiene tutti i libri presenti nel sistema, ognuno dei quali è identificato da un BCID generato univocamente;
- **Prenotazione** : rappreseta la relazione N:N tra Utente e Libro;
- **Passaggio** : contiene la sequenza di utenti che dovrebbero partecipare in maniera attiva alla prenotazione indicata dalla chiave primaria della tabella stessa;
- **Possesso** : questa tabella rappresenta i libri attualmente in possesso dagli utenti iscritti alla community.

Part II

Iterazione 1

4

Architettura

4.1 Architettura software

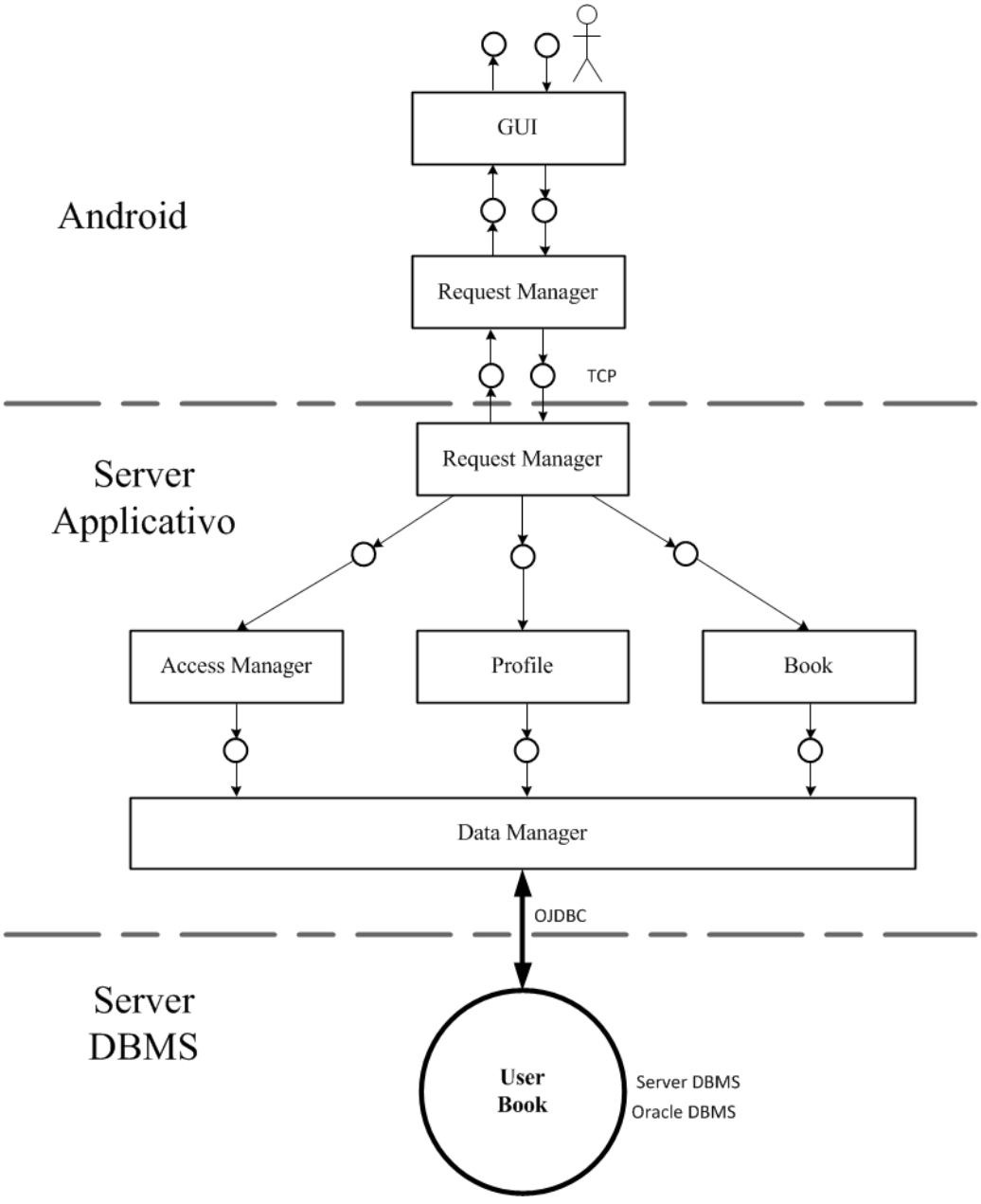

Figure 4.1: Architettura Software

Nella figura 4.1 è mostrata l'architettura software modellizzata attraverso una rete di Petri.

Innanzitutto si può già osservare come essa sia stata definita seguendo il modello architetturale MVC:

- A monte dell'intera applicazione è prevista una parte riservata all'interfaccia grafica, attraverso la quale sarà possibile inviare e ricevere informazioni dal server applicativo. Si vede, infatti, che è stata predisposta una comunicazione bidirezionale tra dispositivo Android e Server AWS.
- Al centro sono rappresentate tutte le richieste a cui il lato server è in grado di rispondere, ovvero funzioni implementate lato Server.
- Infine è prevista una banca dati persistente, in questo caso un database relazionale, al quale il Server Applicativo accede sia per operazioni di lettura che di scrittura, sempre con lo scopo di far fronte alle richieste provenienti dal lato utente.

Si può quindi constatare che non si trattano di strati tra loro indipendenti, poichè il flusso dei dati li coinvolge tutti.

4.2 Logical view

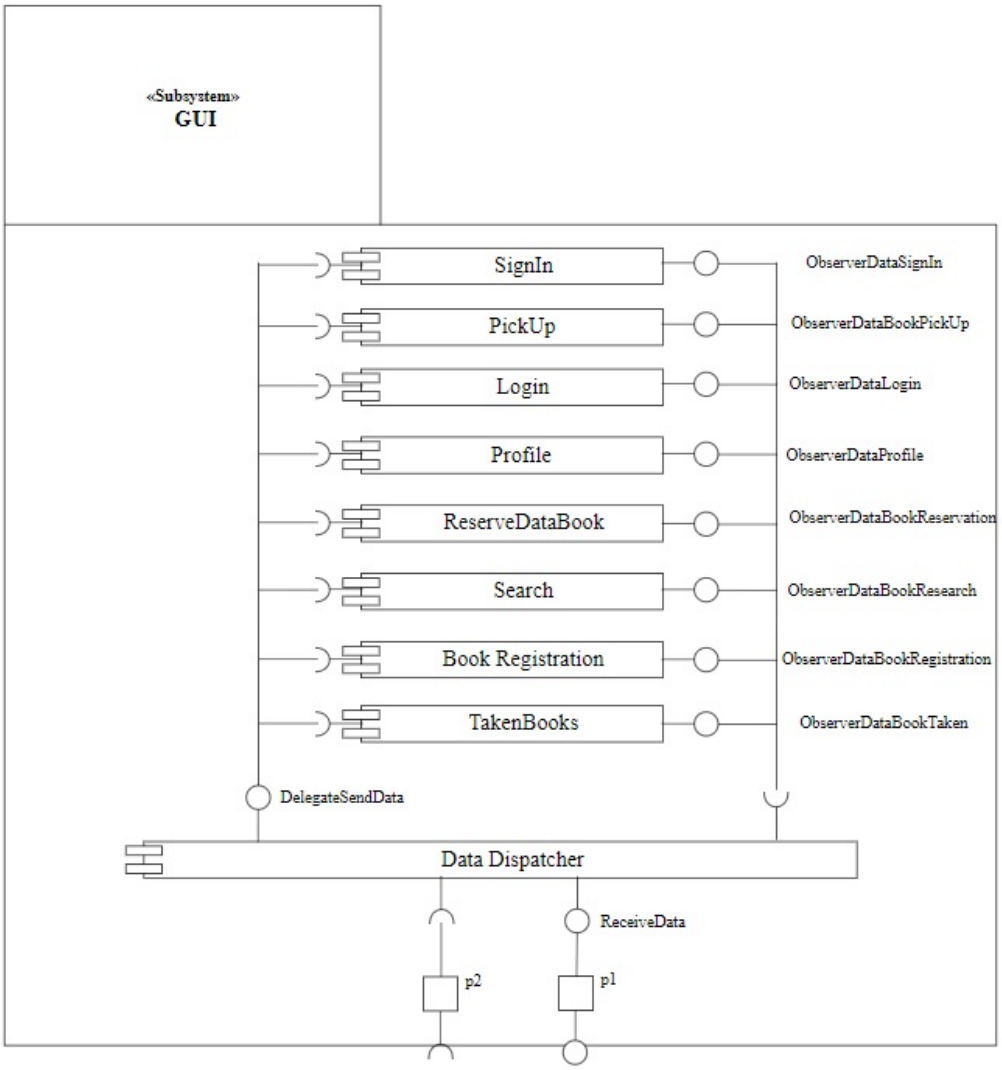

Figure 4.2: Logical view - GUI subsystem

Nelle figure 4.2, 4.3, 4.4 e 4.5 è mostrata in dettaglio la Logical View del sistema progettato.

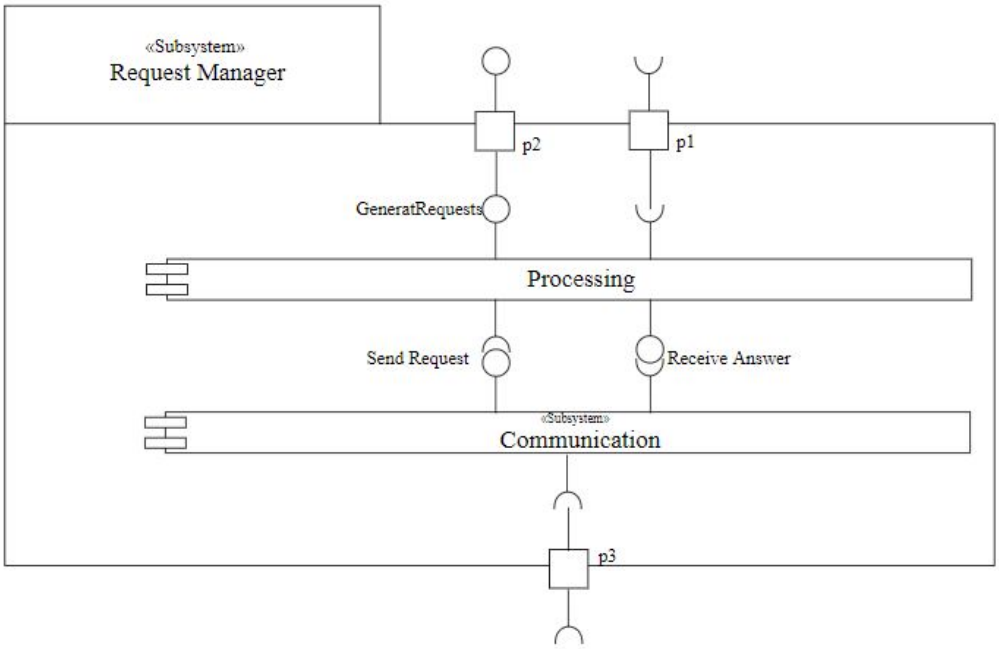

Figure 4.3: Logical view - Request Manager subsystem client side

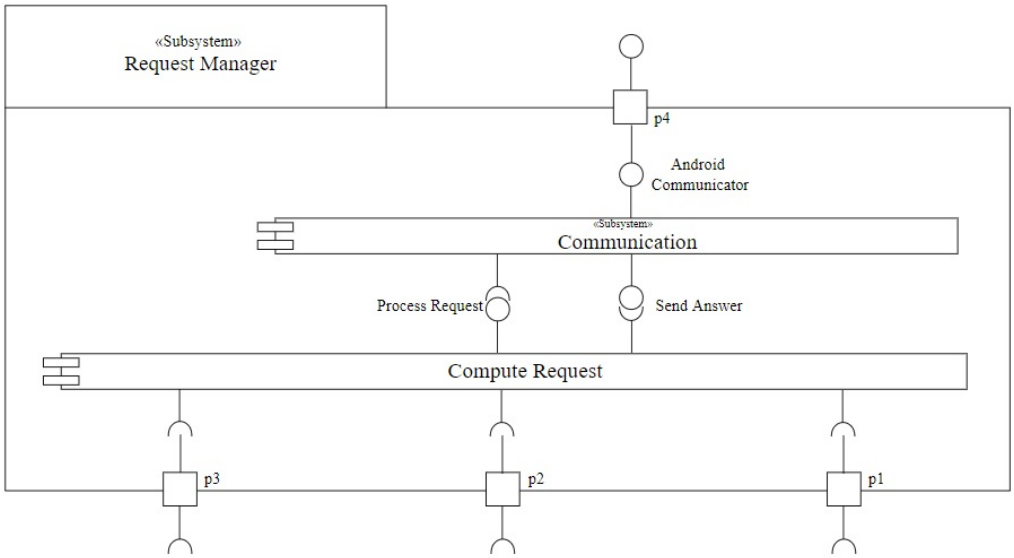

Figure 4.4: Logical view - Request Manager subsystem server side

Si può osservare che, come già introdotto in precedenza, essa segue il modello definito attraverso il pattern archietteturale Model View Controller, dal momento che vengono individuati tre strati, ciascuno dei quali presenta le seguenti caratteristiche:

- **Subsystem *"GUI":*** rappresenta l'interfaccia grafica con la quale l'applicazione si presenterà. Ciascun componente fa riferimento ad ogni view che l'applicazione può mostrare e che quindi corrispondono a diffenti casi d'uso dell'applicativo stesso, come per esempio l'accesso alla rete di Book Crossing (Login) o la registrazione di un libro.

  Questi componenti saranno quindi ovviamente allocati direttamente sul dispositivo mobile: ogni singolo component (*fragment*) avrà legato ad esso, in maniera intrinseca, anche un file con estensione *.xml*, il quale permette di definirne l'interfaccia grafica, ovvero il posizionamento degli elementi visivi.

- **Subsystem *"Request Manager":*** ha il compito di gestire le richieste provenienti da ciascun

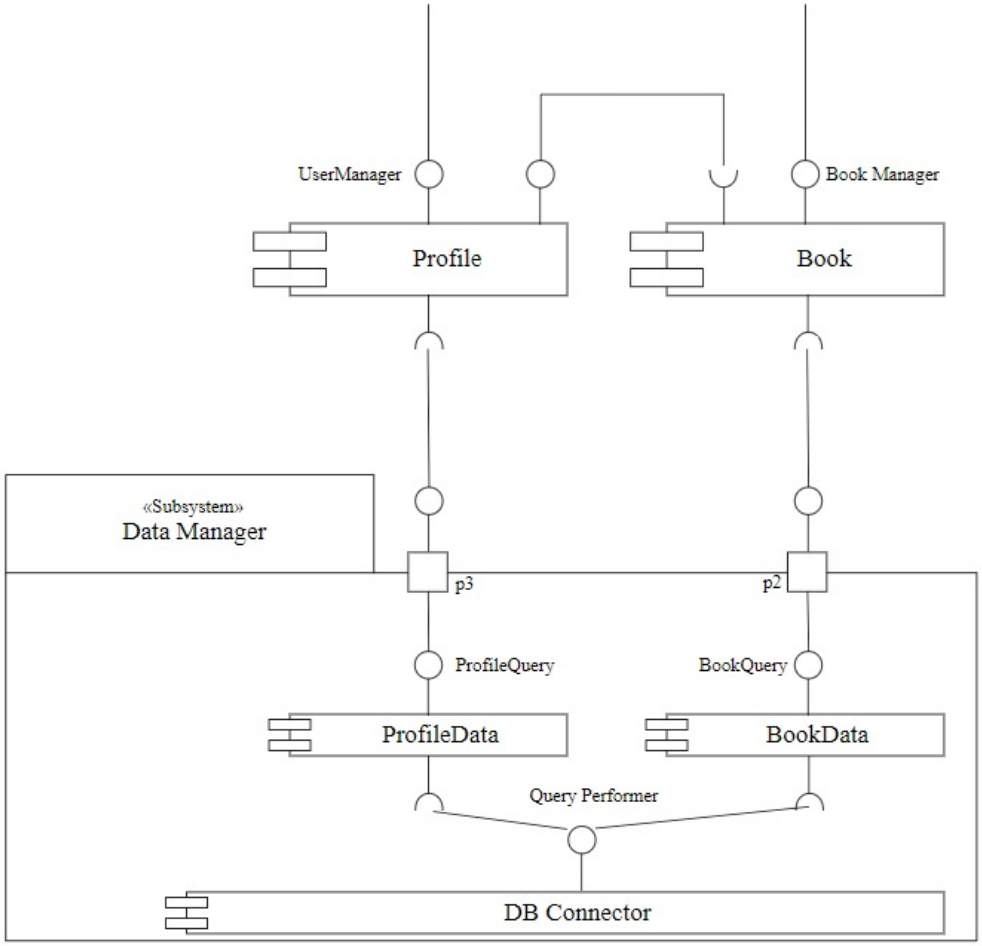

Figure 4.5: Logical view - Server functionality

componente descritto nel subsystem *"GUI"*; al suo interno sono indicati i componenti attraverso i quali si risponde alle richieste provenienti dal dispositivo mobile.

Una parte di questo *manager* sarà disposta a bordo del dispositivo, permettendo una preelaborazione e assemblamento delle richieste; questo subsystem avrà però anche un implementazione *server side*, che gli permetterà di ricevere tali richieste, rieditarle, secondo quello che sono le necessità delle richieste stesse e poi interfacciarsi, in un verso o nell'altro, con la parte di persistenza dei dati.

- **Subsystem *"Data Manager":*** rappresenta la comunicazione con il Database. Sono quindi indicati i componenti utilizzati dal sistema per interfacciarsi con la banca dati dell'architettura.

Si vede quindi come ogni parte dell'architettura abbia un compito ben definito: la parte relativa al subsystem *GUI (Graphic User Interface)* si occupa di gestire l'interazione con l'utente ricevendo e/o mostrando i dati forniti e/o richieste dall'utilizzatore stesso; al suo interno quindi non trovermo codice di logica applicativa ma solamente componenti di gestione *UI*.

Esso rappresenta quindi la parte di ***view***.

Il subsystem *request manager* invece si occupa di controllare il flusso di dati dall'applicazione al server e viceversa; rappresenta quindi la sezione di ***controller*** dove è contenuta la *low logic* dell'applicazione.

Infine, il terzo ed ultimo componente della struttura MVC, è rappresentato dal subsystem *data manager*, il quale permette di interfacciarsi direttamente con la base di dati, astrendo tutte le operazione di controllo di accesso al database stesso (***model***).

Il modello architetturale MVC è stato poi applicato anche successivamente per la progettazione delle componenti previste per ciascun elemento dell'architettura.

5

Analisi dei componenti

5.1 Scelta delle funzionalità da implementare

Come già presentato in precedenza, per la definizione dei componenti si è deciso di seguire il pattern architetturale MVC. Le componenti che si è deciso di sviluppare durante la prima iterazione sono:

- **Componente *Manual Book registration:*** la componente *Manual Book registration* fa riferimento all' UC9 (2.1), figlio del caso d'uso più generico UC5 (2.1), ovvero alla funzione di aggiunta di un libro alla rete di Book Crossing per via manuale.

  La componente si presenta nel seguente modo:

  - *GUI:* interfaccia grafica utilizzata per registrare un libro alla rete di Book Crossing. Verranno quindi messe a disposizione una serie di interfaccie grafiche, composte sostanzialmente da campi da compilare, per aggiungere le informazioni relative al proprio libro, ottenendo poi, successivamente alla registrazione, il relativo BCID;
  - *Model:* si fa carico di ricevere le informazioni relative al libro e, sfruttando la parte *Data*, restituisce alla parte *GUI* il BCID con il quale siglare il libro;
  - *Data:* le informazioni relative al libro che si vuole aggiungere sono memorizzate nel Database RDS, associandolo all'utente che attualmente lo possiede.

- **Componente *Ricerca:*** la componente di *Ricerca* fa riferimento all' UC6, ovvero alla funzione che permette di andare a ricercare un libro all'interno della piattaforma di Book crossing. Questa ricerca può avvenire per titolo, per autore oppure sia per autore che per titolo. Il componente si presenta nel seguente modo:

  - *GUI:* interfaccia grafica composta da due textbox in cui andare ad inserire titolo e/o autore. Essendo possibili tre tipologie di ricerca, come specificato in precedenza, non è necessario compilare entrambi i campi (lo è solamente nel caso in cui si è interessati a compiere una ricerca basandosi su entrambi i vincoli);
  - *Model:* ad esso compete la parte relativa allo smistamento delle richieste, a seconda del fatto che si stia eseguendo una ricerca per titolo, autore o per entrambi;
  - *Data:* fornisce, se presenti, le informazioni relative al libro oggetto della ricerca.

5.2 Aderenza UML a Logical View

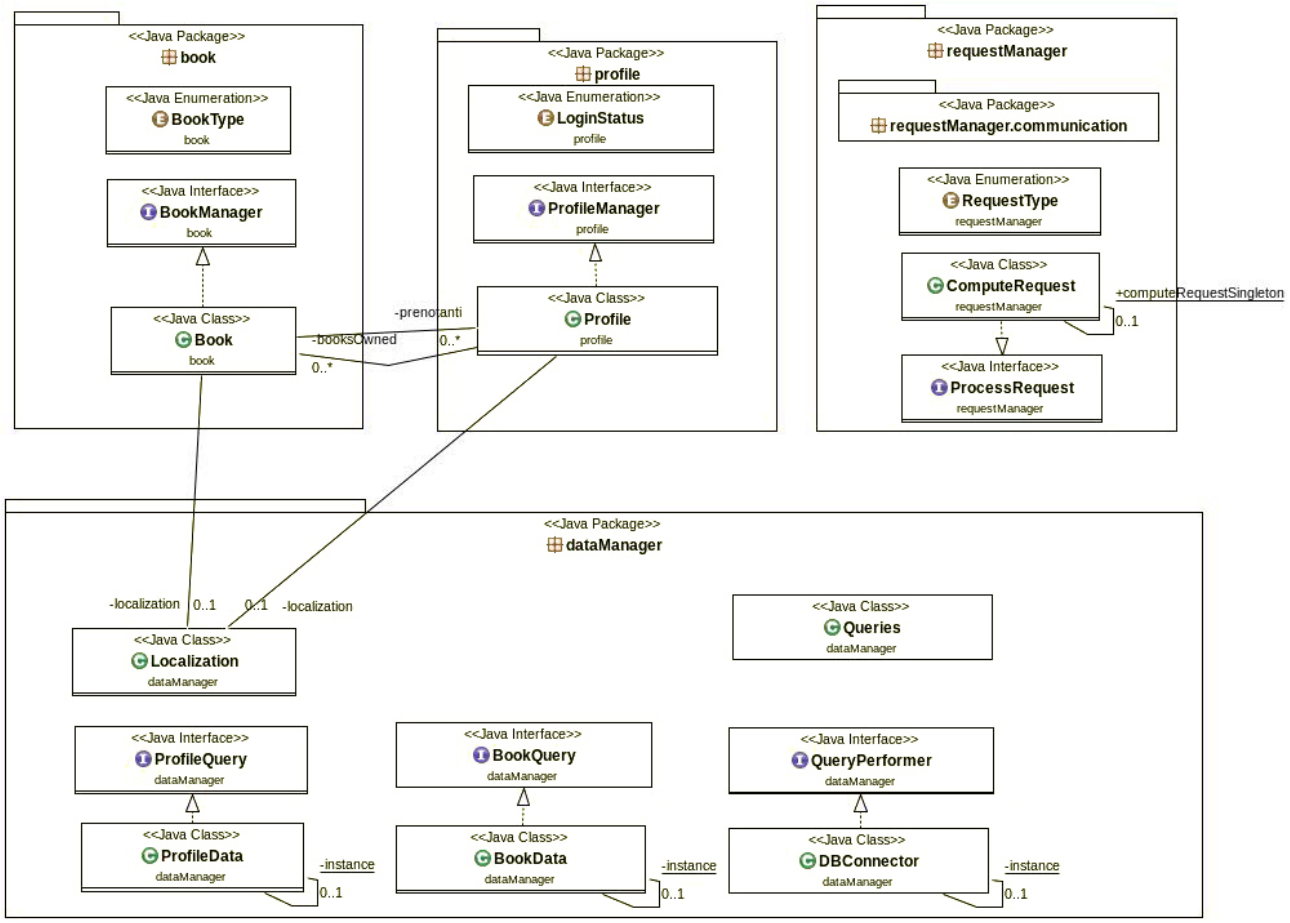

Figure 5.1: Class diagram

```java
public class ComputeRequest implements ProcessRequest {
  public final void process(String msg, String username) {
  // First information is the request type, that  is encoded in this way: //
    REQUESTTYPE:...;
    int i = msg.indexOf(";", 0);
    int j = msg.indexOf(":", 0);
    // Convert the String information of the request type to an enum
    @Nullable RequestType requestType = RequestType.getEnumReqType(msg.substring(0, i
    ).substring(j+1));

    if(requestType == null) {
      // In this case, in the request, there's a wrong encoded request type
      Communication.getInstance().send(username,"requestType:" + 10000 + ";result:
    KO_RequestType");
      return;
    }

    switch(requestType) {
      case BOOK_REGISTRATION_MANUAL:
        Book b = new Book(msg.substring(i + 1));
        b.setActualOwnerUsername(username);
        b.setISBN("null");

        boolean result = b.insert();
        Communication.getInstance().send(username,"requestType:0;result:" + (result?
    1:0) + ";BCID:" + b.getBCID());
        break;
```

```
24
25        case BOOK_SEARCH:
26          if(type.equals("TITLE")) {
27            books = BookData.searchBookByTitle(title);
28          } else if (type.equals("AUTHOR")) {
29            books = BookData.searchBookByAuthor(author);
30          } else {
31            books = BookData.searchBook(title, author);
32          }
33          Communication.getInstance().send(username,"requestType:8;result:" + 1 + ";
    Books:" + jsonResponse);
34          break;
35       }
36     }
37   }
```

TODO: specificare meglio cosa intendiamo dire qua

Analizzando le componenti presenti nel codice e quelle schematizzate all'interno dell'UML, possiamo osservare come la classe ComputeRequest si preoccupi di gestire le richieste gestite bella prima iterazione provenienti dal client nel formato descritto.

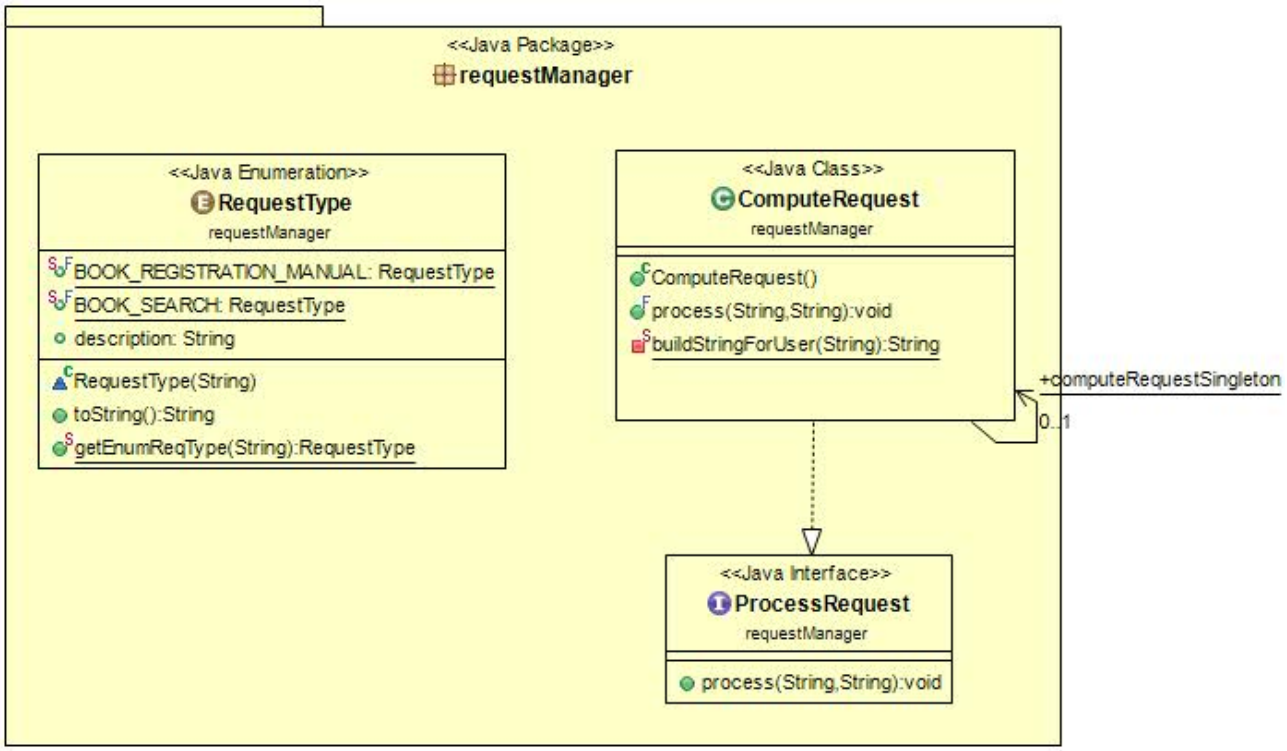

Figure 5.2: Diagramma della classe ComputeRequest

6

Observer-delegate pattern: la nostra implementazione

L'applicativo, oltre a seguire un design MVC per la separazione e l'organizzazione dei componenti, implementa anche un pattern di comunicazione ad eventi, meglio conosciuto come *pattern observer-delegate*.

In sostanza, ogni componente che necessita di una certa tipologia di informazioni si mette in ascolto registrandosi ad un "servizio" offerto, nel nostro caso, dal *DataDispatcherSingleton* il quale va a salvarsi al suo interno un gruppo di riferimenti (delegati) che dovrà poi notificare nel momento in cui risulteranno essere presenti nuove informazioni a cui gli observers risultano essere interessati.

Come si può ben capire si tratta quindi di un pattern molto vicino alla struttura *Publish-Subscriber*, con il quale si definisce una dipendenza uno a molti fra gli oggetti: in sostanza abbiamo un oggetto che viene *"osservato"* e tanti oggetti che *"osservano"* i cambiamenti di quest'ultimo, come si può vedere nella struttura generica rappresentata in figura 6.1.

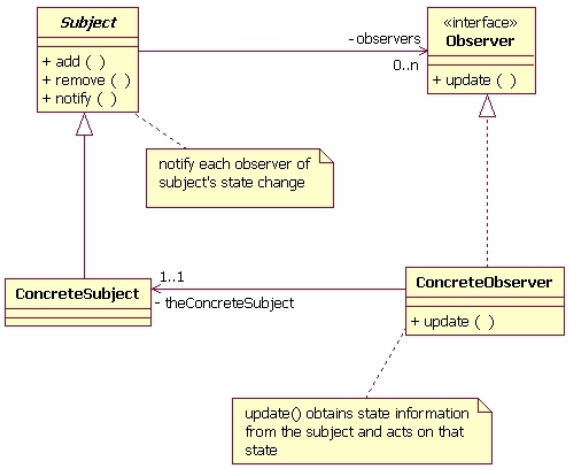

Figure 6.1: Struttura generica pattern observer

Questa struttura è stata riprodotta anche nel nostro applicativo, dove è possibile notare la presenza di alcune strutture caratteristiche di questo pattern: abbiamo infatti una funzionalità di *register*, che, a seconda di quale observer richiama questa funzione, pone l'oggetto nel vettore specifico (figura 6.2). Per evitare di avere dipendenze cicliche o comunque per mantenere una certa pulizia nei riferimenti tra classi, è necessario che ogni observer, nel momento in cui non necessita più di una certa tipologia di informazionw, vada a deregistrarsi: questa operazione consiste nell'andare a togliere qualsiasi riferimento dell'oggetto in esame dall'interno del *DataDispatcherSingleton*, come fatto in figura 6.3.

Per quanto riguarda invece la parte di *notifica* abbiamo introdotto un pattern ad eventi custom,mantenendo così separata l'implementazione delle informazioni ottenuto da ogni singolo fragment, come si vede nella figura 6.4.

```
@Override
public void register(ObserverForUiInformation observerForUiInformation) {
    if ((observerForUiInformation instanceof ObserverDataBookRegistration) && (!observersDataBookRegistration.contains(observerForUiInformation))) {
        observersDataBookRegistration.add((ObserverDataBookRegistration) observerForUiInformation);
    } else if ((observerForUiInformation instanceof ObserverDataBookPickUp) && (!observersDataBookPickUp.contains(observerForUiInformation))) {
        observersDataBookPickUp.add((ObserverDataBookPickUp) observerForUiInformation);
    } else if ((observerForUiInformation instanceof ObserverDataBookTaken) && (!observersDataBookTaken.contains(observerForUiInformation))) {
        observersDataBookTaken.add((ObserverDataBookTaken) observerForUiInformation);
    } else if ((observerForUiInformation instanceof ObserverDataLogin) && (!observersDataLogin.contains(observerForUiInformation))) {
        observersDataLogin.add((ObserverDataLogin) observerForUiInformation);
    } else if ((observerForUiInformation instanceof ObserverDataSignIn) && (!observersDataSignIn.contains(observerForUiInformation))) {
        observersDataSignIn.add((ObserverDataSignIn) observerForUiInformation);
    } else if ((observerForUiInformation instanceof ObserverDataProfile) && (!observersDataProfile.contains(observerForUiInformation))) {
        observersDataProfile.add((ObserverDataProfile) observerForUiInformation);
    } else if ((observerForUiInformation instanceof ObserverDataBookResearch) && (!observarDataBookResearch.contains(observerForUiInformation))) {
        observarDataBookResearch.add((ObserverDataBookResearch) observerForUiInformation);
    } else if ((observerForUiInformation instanceof ObserverDataBookReservation) && (!observarDataBookReservation.contains(observerForUiInformation))) {
        observarDataBookReservation.add((ObserverDataBookReservation) observerForUiInformation);
    }
}
```

Figure 6.2: Register function

```
@Override
public boolean unRegister(ObserverForUiInformation observerForUiInformation) {
    if (observerForUiInformation instanceof ObserverDataBookRegistration) {
        return observersDataBookRegistration.remove(observerForUiInformation);
    } else if (observerForUiInformation instanceof ObserverDataBookPickUp) {
        return observersDataBookPickUp.remove(observerForUiInformation);
    } else if (observerForUiInformation instanceof ObserverDataBookTaken) {
        return observersDataBookTaken.remove(observerForUiInformation);
    } else if (observerForUiInformation instanceof ObserverDataLogin) {
        return observersDataLogin.remove(observerForUiInformation);
    } else if (observerForUiInformation instanceof ObserverDataProfile) {
        return observersDataProfile.remove(observerForUiInformation);
    } else if (observerForUiInformation instanceof ObserverDataSignIn) {
        return observersDataSignIn.remove(observerForUiInformation);
    } else if (observerForUiInformation instanceof ObserverDataBookResearch) {
        return observarDataBookResearch.remove(observerForUiInformation);
    } else if (observerForUiInformation instanceof ObserverDataBookReservation) {
        return observarDataBookReservation.remove(observerForUiInformation);
    }
    return false;
}
```

Figure 6.3: Unregister function

Ogni singola interfaccia implementa a sua volta un'interfaccia comune a più alto livello: questa scelta è stata adotta per rendere del tutto generico il tipo di observer che va a registrarsi per gli eventi forniti dal delegate.

Infatti, come si può vedere nell'interfaccia *DelegateSendData* (figura 6.5), chi è interessato ad ottenere informazioni specifiche si registra come un oggeto di tipo *ObserverForUiInformation*; sarà poi compito del delegate fornire le corrette informazioni andando a richiamare le corrette funzioni di notifica (direttamente implementate all'interno di ogni singola interfaccia concreta).

Si può quindi vedere che, nella struttura predisposta, il *DataDispatcher* lavora come se fosse un **repository**, ovvero un contenitore di informazioni, in grado di fornire a tutti gli elementi che si registrano i dati di cui necessitano.

Questi elementi, allo stadio implementativo attuale, sono rappresentati da tutti i fragment, che devo ricevere/inviare informazioni per soddisfare l'interazione con l'utente: come si può notare nelle figure 6.6- 6.7- 6.7- 6.9, ogni singolo fragment è predisposto per ricevere le informazioni di cui ha bisogno, tramite la chiamata delle callbacks da parte del dispatcher stesso.

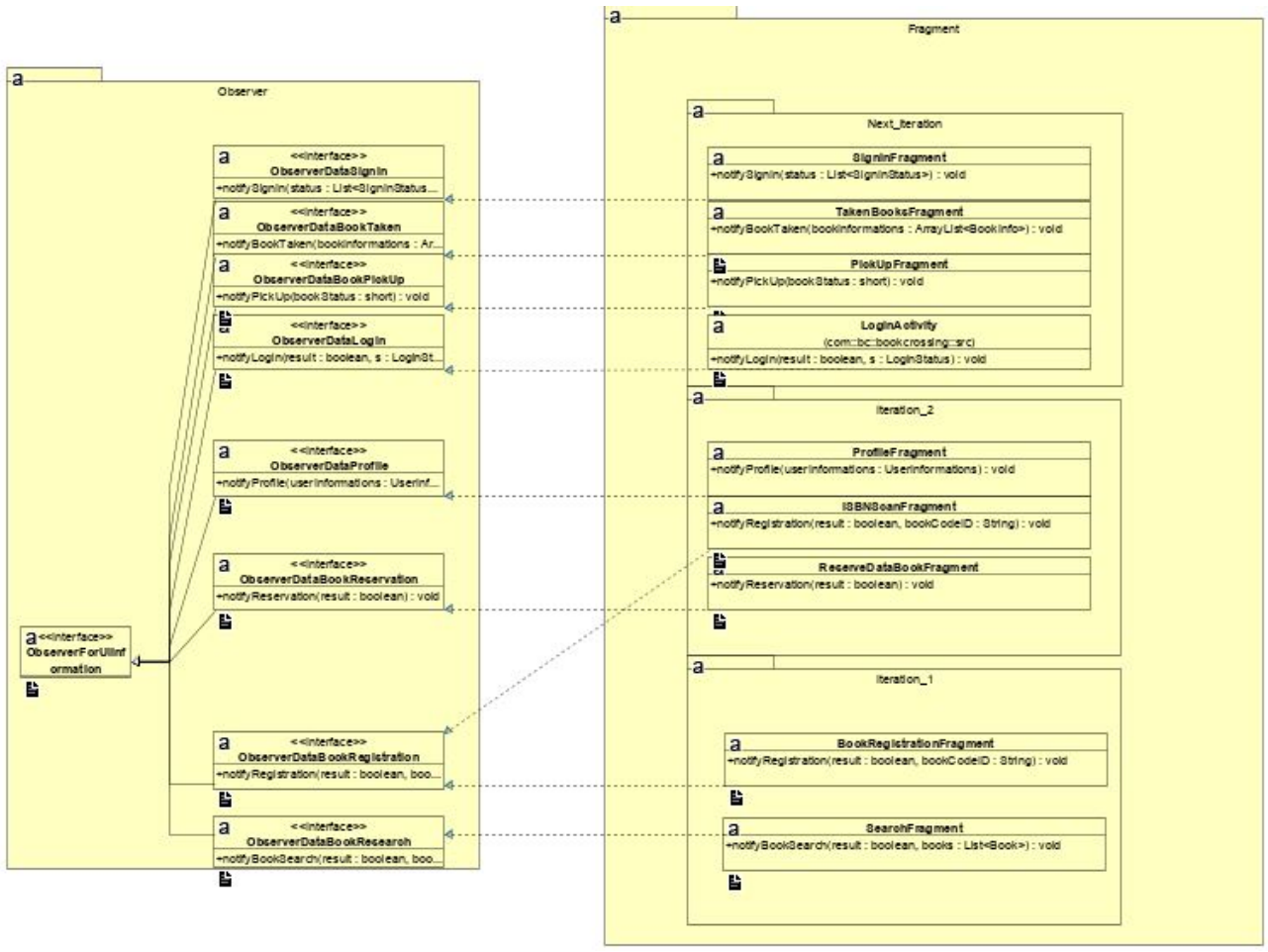

Figure 6.4: Interfaccie per la ricezione degli eventi da lato server

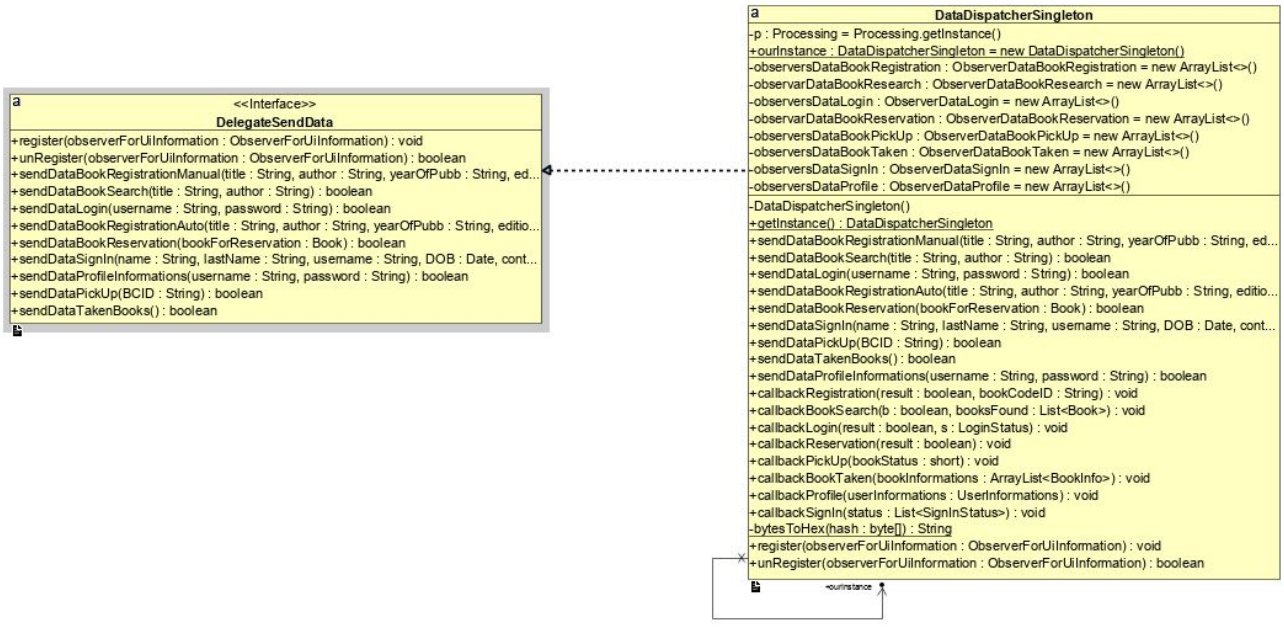

Figure 6.5: Struttura del componente *DataDispatcher*

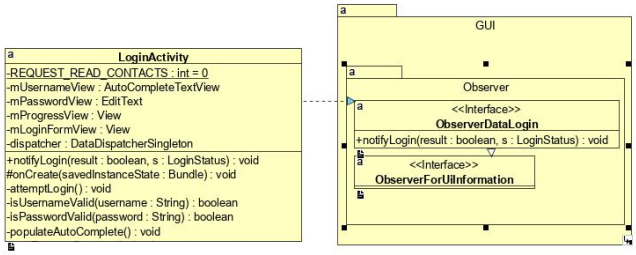

Figure 6.6: Struttura del componente *LoginFragment*

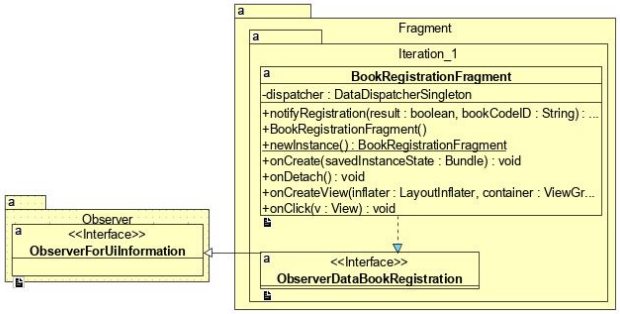

Figure 6.7: Struttura del componente *BookRegistrationFragment*

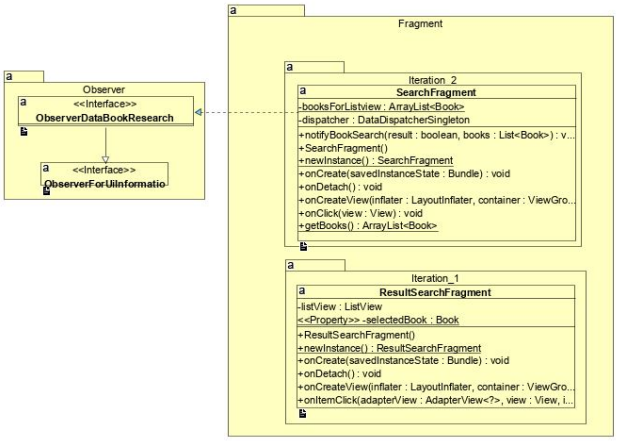

Figure 6.8: Struttura del componente *ResearchFragment*

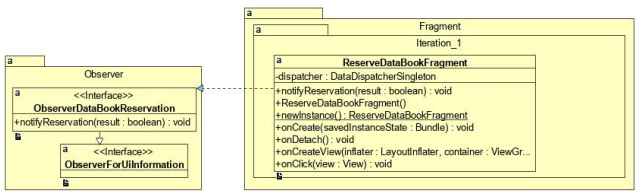

Figure 6.9: Struttura del componente *ReservationFragment*

7

Implementazione comunicazione con server

7.1 Il framework *Netty*

Netty è un framework NIO (Non-blocking Input Output) di tipo client-server (i.e. permette una comunicazione asincrona tra client e server di tipo event-driven) che facilita lo sviluppo di applicazioni di rete che utilizzano socket TCP e UDP.

Il framework è disponibile sul Maven Central Repository e può essere scaricato includendo la seguente dependency nel pom.xml del progetto:

```
<dependencies>
  <dependency>
    <groupId>io.netty</groupId>
    <artifactId>netty-all</artifactId>
    <version>4.1.32.Final</version>
  </dependency>
</dependencies>
```

Listing 7.1: estratto del pom.xml relativo alle dependency

Lato Server creiamo un oggetto singleton di tipo *Communication* il quale inizializza a sua volte un oggetto di tipo *ServerBootstrap* (io.netty.bootstrap.ServerBootstrap).

Quest'ultimo effettua il bootstrap del *ServerChannel*, dove il *ServerChannel* rappresenta il link "primario" (sul port 5000) che accetta ogni connessione di un client e crea un canale "figlio" sul quale avverà tale comunicazione.

```
private Communication() throws Exception {
  ServerBootstrap b = new ServerBootstrap();

  b.group(bossGroup, workerGroup).channel(NioServerSocketChannel.class).handler(new
    LoggingHandler(LogLevel.INFO)).childHandler(new ServerInitializer(sslCtx));

  b.bind(PORT).sync().channel().closeFuture().sync();
}
```

Listing 7.2: definizione costruttore lato server

Al server andiamo ad aggiungere un handler *ServerInitializer* che aggiunge alla pipeline, oltre agli encoder e decoder di default, il *ServerHandler*.

```
52 class ServerInitializer extends ChannelInitializer<SocketChannel> {
53
54   private static final StringDecoder DECODER = new StringDecoder();
55   private static final StringEncoder ENCODER = new StringEncoder();
56   private static final ServerHandler SERVER_HANDLER = new ServerHandler();
57
58   @Override
59   public void initChannel(SocketChannel ch) throws Exception {
60     ChannelPipeline pipeline = ch.pipeline();
61     pipeline.addLast(DECODER);
62     pipeline.addLast(ENCODER);
63     // and then business logic.
64     pipeline.addLast(SERVER_HANDLER);
65   }
66 }
```

Listing 7.3: inizializzazione campi per comunicazione lato server

La classe *ServerHandler* è un sottotipo di *SimpleChannelInboundHandler* (io.netty.channel.SimpleChannelInboundHandler) il quale fornisce la stringa inviata dal client tramite il metodo seguente.

```
67 class ServerHandler extends SimpleChannelInboundHandler<String> {
68
69   @Override
70   public void channelRead0(ChannelHandlerContext ctx, String request) throws
       Exception {
71     int j = request.indexOf(";");
72     String username = request.substring(0, j);
73     Communication.chcMap.put(username, ctx);
74
75     System.out.println("Process request: " + request);
76     computeRequest.process(request.substring(j + 1), username);
77   }
78 }
```

Listing 7.4: parsing e invio del messaggio nel formato desiderato verso client

Si può notare che il metodo precedente memorizza la coppia (username, *channelHandlerContext*) all'interno dell'oggetto chcMap di *Communication*, il quale è di tipo HashMap<String, ChannelHandlerContext>. Tale informazione è fondamentale al fine di rispondere al client tramite canale "figlio" ad esso dedicato e chiuderlo al termine della comunicazione: il metodo invoca poi l'elaborazione del messaggio ricevuto tramite l'interfaccia *ProcessRequest* (la semantica della stringa request verrà discussa nel paragrafo successivo).

Infine, ma non per importanza, discutiamo l'implementazione dell'interfaccia *SendAnswer* da parte della classe *Communication*:

```
79 public final class Communication implements SendAnswer{
80
81   public void send(final String username, final String msg) {
82     ChannelFuture oo = chcMap.get(username).writeAndFlush(msg + "\r\n");
83     oo.addListener(new ChannelFutureListener() {
84
85       public void operationComplete(ChannelFuture future) throws Exception {
86         int count = 0;
87         boolean isError = false;
88
89         while(!future.isSuccess()) {
90           chcMap.get(username).writeAndFlush(msg + "\r\n");
91           System.out.println("Retry send response!");
92           if(count == 3) {
93             isError = true;
94             break;
95           } else {
96             count ++;
97           }
98         }
99
100        future.addListener(ChannelFutureListener.CLOSE);
101        chcMap.remove(username);
102      }
103    });
104  }
105 }
```

Listing 7.5: My Caption

Il metodo Send, definito all'interno della classe *Communication*, si occupa di inviare la risposta relativa alla richiesta ricevuta e, nel caso di fallimento, ripeterà l'invio fino a un limite di 3 tentativi.

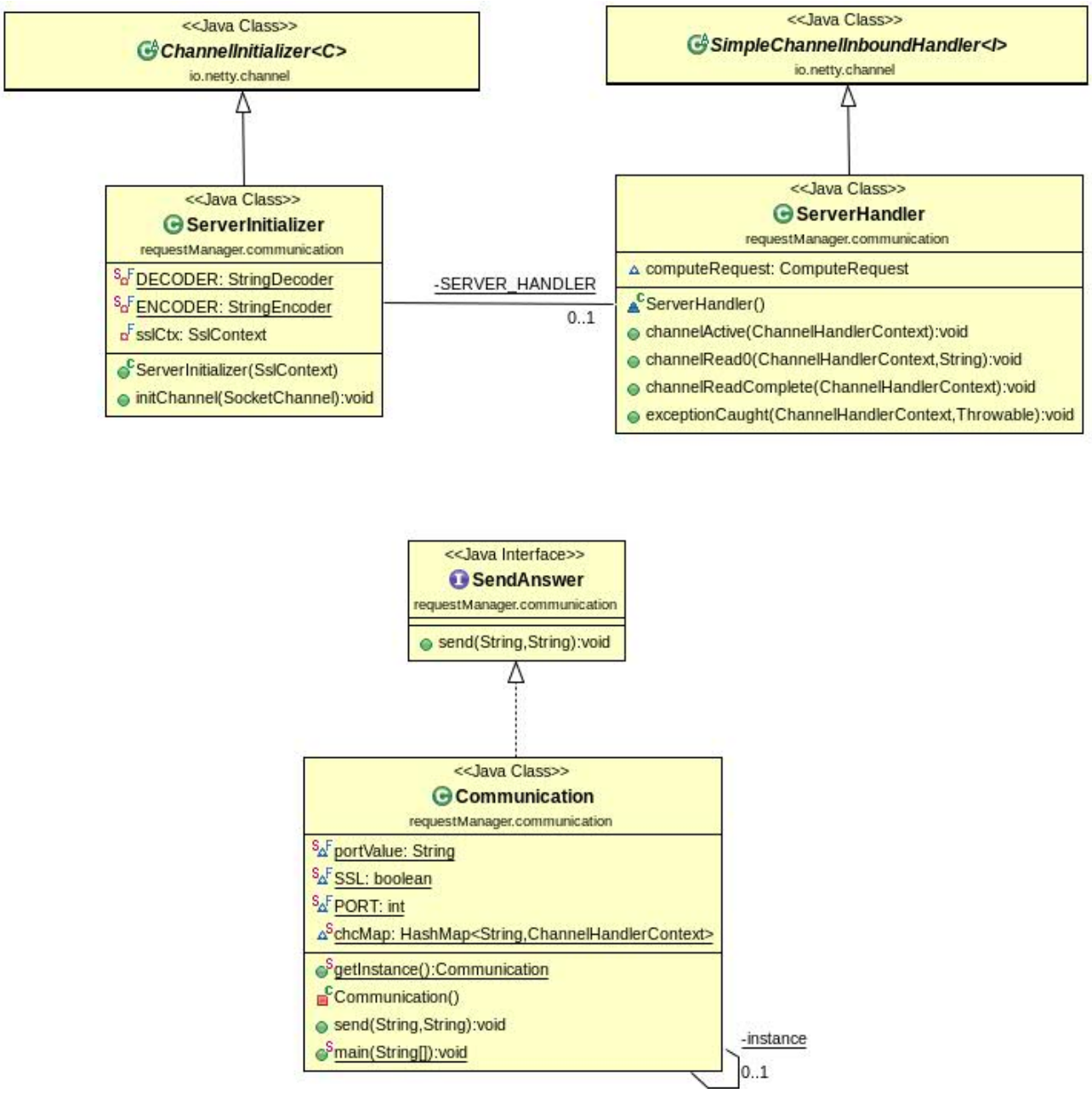

Figure 7.1: Struttura del package *requestManager.Communication*

Lato Client creiamo un oggetto singleton *Communication* che implementa il seguente metodo dell'interfaccia *SendRequest*, aprendo la connessione con il Server sulla socket (35.180.103.132,5000) e inviando il messaggio al server.

```
106 public class Communication implements SendRequest {
107   private static final String IP = "35.180.103.132";
108   public static final String HOST = System.getProperty("host", IP);
109   public static final int PORT = 5000;
110   @Override
111   public boolean send(String data) {
112     Bootstrap b = new Bootstrap();
113     b.group(group).channel(NioSocketChannel.class).handler(new ClientInitializer(
      sslCtx));
114     ChannelFuture ch = null;
115
116     ch = b.connect(HOST, PORT);
117
```

```
118     ChannelFuture lastWriteFuture = null;
119     lastWriteFuture = ch.channel().writeAndFlush(data + "\r\n");
120
121     //Wait until all messages are flushed before closing the channel
122     if (lastWriteFuture != null) {
123       lastWriteFuture.sync();
124     }
125   }
126 }
```

Listing 7.6: Connessione con server e invio del messaggio

Tralasciando la discussione dell'implementazione della classe *ClientInitializer* in quanto simile a quella della classe duale lato server.

Andiamo invece ad approfondire l'implementazione della classe *ClientHandler*:.

```
127 class ClientHandler extends SimpleChannelInboundHandler<String> {
128   @Override
129   protected void channelRead0(ChannelHandlerContext ctx, String msg) {
130     Processing.getInstance().processAnswer(msg);
131     Communication.getInstance().group.shutdownGracefully();
132   }
133 }
```

Listing 7.7: lettura di quanto inviato dal client

Di particolare interesse è il metodo *channelRead0* ereditato da *SimpleChannelInboundHandler*, il quale viene invocato dal framework ogni qual volta il client riceve un messaggio.

Il messaggio viene poi elaborato dal oggetto singleton *Processing* e il canale di comunicazione viene chiuso.

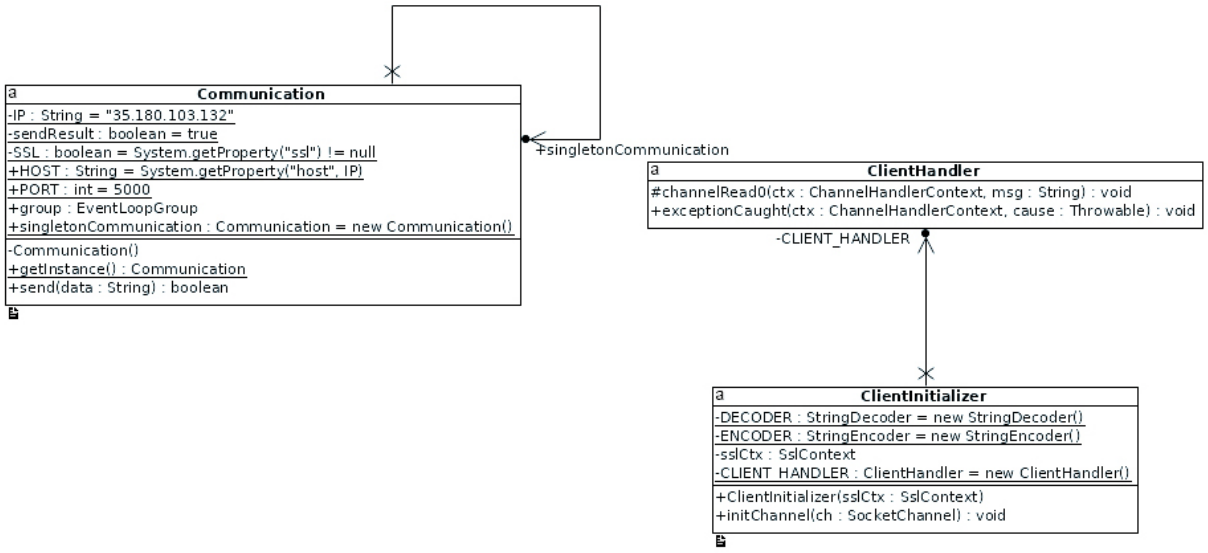

Figure 7.2: Struttura del package *requestManager.Communication*

7.2 Semantica dei messaggi

I messaggi inviati dai client, ovvero direttamente dagli users, non sono altro che stringhe le quali possono essere identificate dalla seguente *RegEx* (*Regular Expression*):

```
134 ^[a-zA-Z0-9]+;[a-zA-Z0-9]+:[0-9]+;([a-zA-Z0-9]+:[a-zA-Z0-9]+;)+$
```

Listing 7.8: *RegEx* utilizzata

la cui semantica può essere così rappresentata:

"<username>;requestType:<tipo>;<richiesta>"

I valori assunti da <tipo> sono i seguenti e corrispondo ai campi definiti all'interno dell'enumerativo *RequestType*.

Durante la prima iterazione siamo andati a gestire le richieste relative alle sole due seguenti tipologie di richieste:

- 0 -> BOOK_REGISTRATION_MANUAL
- 8 -> BOOK_SEARCH

Nella seconda iterazione, e nelle successive, è stata pianificata l'implementazione della parte di gestione delle restanti richieste:

- 1 -> BOOK_RESERVATION
- 2 -> LOGIN
- 3 -> SIGN_IN
- 4 -> BOOK_REGISTRATION_AUTOMATIC
- 5 -> PROFILE_INFO
- 6 -> TAKEN_BOOKS
- 7 -> PICK_UP

La richiesta inviata dal client Android verso il server, ad esempio, per la ricerca di un libro assume la seguente forma:

```
135 username + ";" + "requestType:" + 0 + ";" + book.encode();
```

Listing 7.9: Semantica della richiesta per ricerca

Lato Android la richiesta vera e propria è preceduta dall'username dell'utente collegato in modo che, lato server, qualora per diversi motivi l'username fosse invalido, la richiesta venga ignorata a priori. Analogamente possiamo identificare i messaggi inviati dal server come:

```
136 ^[a-zA-Z0-9]+:[0-9]+;([a-zA-Z0-9]+:[a-zA-Z0-9]+;)+$
```

Listing 7.10: *RegEx* utilizzata lato server per l'interpretazione delle richieste

la cui semantica può a sua volta essere così rappresentata:

"requestType:<tipo>;<risultato>"

Ad esempio, nel caso di invio di una risposta da parte del server verso il client Android, in merito alla registrazione manuale di un libro andata a buon fine, la stringa va ad assumere il seguente formato:

```
137  "requestType:0;result:" + 1 + ";BCID:" + bcid
```

Listing 7.11: Risposta OK

dove *bcid* è il codice alfanumerico generato casualmente dal server per identificare il libro all'interno della rete di Book Crossing.

Qual'ora invece il server riceva una richiesta con un tipo errato, la risposta che andrebbe ad inoltrare sarebbe la seguente:

```
138  "requestType:10000;result:KO_RequestType"
```

Listing 7.12: Risposta KO

Un possibile sviluppo futuro potrebbe essere quello di sfruttare una semantica con un formato *tradizionale* come JSON.
Il fatto che in questo caso si è scelto un linguaggio proprietario è stato dovuto al fatto che si voleva rendere l'applicazione indipendente da questo punto di vista, in maniera tale che client e server fossero anch'essi proprietari dal punto di vista della semantica del linguaggio. Sicuramente il fatto di sfruttare JSON oppure XML, da un punto di vista della codifica sarebbe risultato più semplice, dal momento che esistono classi che garantiscono il loro parsing in funzione di un valore chiave con il quale viene identificato ogni elemento del messaggio.
In questo caso, invece, la procedura di parsing della stringa del messaggio è risultata più lunga, in quanto si è dovuto, dopo aver stabilito il formato descritto, valutare la posizione dei simboli all'interno della stringa per poterla decodificare.

Part III

Iterazione 2

8

Creazione della base di dati

Per il funzionamento del software è necessario avere su di un Database Server, in questo caso RDS, il database contenente le tabelle per la memorizzazione delle informazioni.

Le query utilizzate per la generazione del database sono le seguenti:

```
-- tables
-- Table: Libro
CREATE TABLE Libro (
  BCID varchar2(15)  NOT NULL,
  Titolo varchar2(256)  NOT NULL,
  Autore varchar2(256)  NOT NULL,
  DataPubblicazione VARCHAR2(20 BYTE)  NULL,
  ISBN varchar2(13),
  Proprietario varchar2(20)  NOT NULL,
  Genere varchar2(256),
  Longitudine number(38,0),
  Latitudine number(38,0),
  CONSTRAINT Libro_pk PRIMARY KEY (BCID)
) ;

-- Table: Possesso
CREATE TABLE Possesso (
  Username varchar2(20)  NOT NULL,
  bcid varchar2(15)  NOT NULL,
  DataInizio date  NOT NULL,
  DataFine date,
  LuogoRilascio varchar2(512),
  CONSTRAINT Possesso_pk PRIMARY KEY (Username,bcid)
) ;

-- Table: Prenotazione
CREATE TABLE Prenotazione (
  Username varchar2(20)  NOT NULL,
  bcid varchar2(15)  NOT NULL,
  ID number(38, 0) NOT NULL,
  CONSTRAINT Prenotazione_pk PRIMARY KEY (ID)
) ;

-- Table: Utente
CREATE TABLE Utente (
  Username varchar2(20)  NOT NULL,
  Nome varchar2(50)  NOT NULL,
  Cognome varchar2(50)  NOT NULL,
  DataNas date  NOT NULL,
  Contatto varchar2(256)  NULL,
  Password char(64)  NOT NULL,
  ResidenzaLat integer  NOT NULL,
  ResidenzaLong integer  NOT NULL,
  RaggioAzione integer  NOT NULL,
  CONSTRAINT Utente_pk PRIMARY KEY (Username)
) ;
```

```
185
186 -- Table: Passaggio
187 CREATE TABLE Passaggio (
188   Utenti varchar2(100) NOT NULL,
189   ID number(38, 0) NOT NULL,
190   CONSTRAINT Passaggio_pk PRIMARY KEY (ID)
191 );
192
193 -- foreign keys
194 -- Reference: Libro_Utente (table: Libro)
195 ALTER TABLE Libro ADD CONSTRAINT Libro_Utente
196 FOREIGN KEY (Propietario)
197 REFERENCES Utente (Username);
198
199 -- Reference: Possiede_Libro (table: Possesso)
200 ALTER TABLE Possesso ADD CONSTRAINT Possiede_Libro
201 FOREIGN KEY (bcid)
202 REFERENCES Libro (BCID);
203
204 -- Reference: Possiede_Profilo (table: Possesso)
205 ALTER TABLE Possesso ADD CONSTRAINT Possiede_Profilo
206 FOREIGN KEY (username)
207 REFERENCES Utente (Username);
208
209 -- Reference: Prenotazione_Libro (table: Prenotazione)
210 ALTER TABLE Prenotazione ADD CONSTRAINT Prenotazione_Libro
211 FOREIGN KEY (bcid)
212 REFERENCES Libro (BCID);
213
214 -- Reference: Prenotazione_Utente (table: Prenotazione)
215 ALTER TABLE Prenotazione ADD CONSTRAINT Prenotazione_Utente
216 FOREIGN KEY (username)
217 REFERENCES Utente (Username);
218
219 ALTER TABLE Passaggio ADD CONSTRAINT Passaggio_Prenotazione
220 FOREIGN KEY (ID)
221 REFERENCES Prenotazione (ID);
222
223 -- End of file.
```

9

Analisi dei componenti

Le componenti che invece si è deciso di andare a sviluppare durante la seconda iterazione sono:

- **Componente *Automatic Book registration:*** la componente *Automatic Book registration* fa riferimento all' UC10 (figlio del caso d'uso più generico UC5); questa funzionalità consente di ottenere, in maniera del tutto automatica, le informazioni necessarie per registrare uno specifico libro all'interno della piattaforma di book crossing. La componente si presenta nel seguente modo:
  - *GUI:* interfaccia grafica utilizzata per registrare un libro, in maniera automatica, nella rete di Book Crossing. Verranno quindi messe a disposizione una serie di interfaccie grafiche, le quali permetteranno di scansionare il codice ISBN del libro deisderato, unitamente ad una parte GUI utilizzata per mostrare le informazioni relative al libro che è appena stato scansionato.
  - *Model:* si fa carico di ricevere le informazioni relative al libro e, sfruttando la parte Data, restituisce alla parte GUI il BCID con il quale siglare il libro;
  - *Data:* le informazioni relative al libro che si vuole aggiungere sono memorizzate nel Database RDS, associandolo all'utente che attualmente lo possiede.
- **Componente *Login:*** la componente *Login* fa riferimento all' UC13 , ovvero alla funzione che permette ad un utilizzatore dell'applicazione di loggarsi all'interno della piattaforma di Book Crossing, potendo così effettuare operazioni di suo interesse sui testi disponibili. La componente si presenta nel seguente modo:
  - *GUI:* interfaccia grafica, composta da due caselle di testo, le quali devono essere riempite con username e password, unitamente ad un pulsante che permette di iniviare la richiesta/verifica di login corretto. Per chi non fosse già registrato, è fornita la possibilità di iscriversi alla piattaforma (questo però rappresenta un caso d'uso separato);
  - *Model:* si fa carico di verificare la coerenza dei dati inseriti tramite il componente grafico, restituendo l'esito a chi ha appena tentato di effettuare il login.
  - *Data:* le informazioni relative all'utente che sta tentando di loggarsi.
- **Componente *Book reservation:*** La componente *Book reservation* fa riferimento all' U13 (2.1), ovvero alla funzione di prenotazione di un libro, la quale richiede prima un'operazione di ricerca dello stesso all'interno della community e poi, se possibile, permette di effettuare la prenotazione effettiva. La componente si presenta nel seguente modo:
  - *GUI:* interfaccia grafica utilizzata per poter procedere con la prenotazione di un libro della rete di Book Crossing. In questo caso sarà possibile compiere tale azione attraverso una procedura di ricerca del libro, oppure navigando nella propria sezione personale del profilo;
  - *Model:* si fa carico di gestire la coda relativa alla prenotazione di un determinato libro, in modo da poterle soddisfare. La gestione e la computazione della prenotazione è affidata

ad un algoritmo il quale va ad appoggiarsi alla parte Data per poter risalire ai dati dei richiedenti;

– *Data:* le informazioni relative al libro che si vuole prenotare e agli utenti richiedenti le quali sono memorizzate nel Database.

Partendo dalla figura 5.2 relativa a quanto sviluppato nella precedente iterazione, possiamo andare ad osservare come, nella successiva immagine, il medesimo diagramma delle classi si completa nel seguente modo:

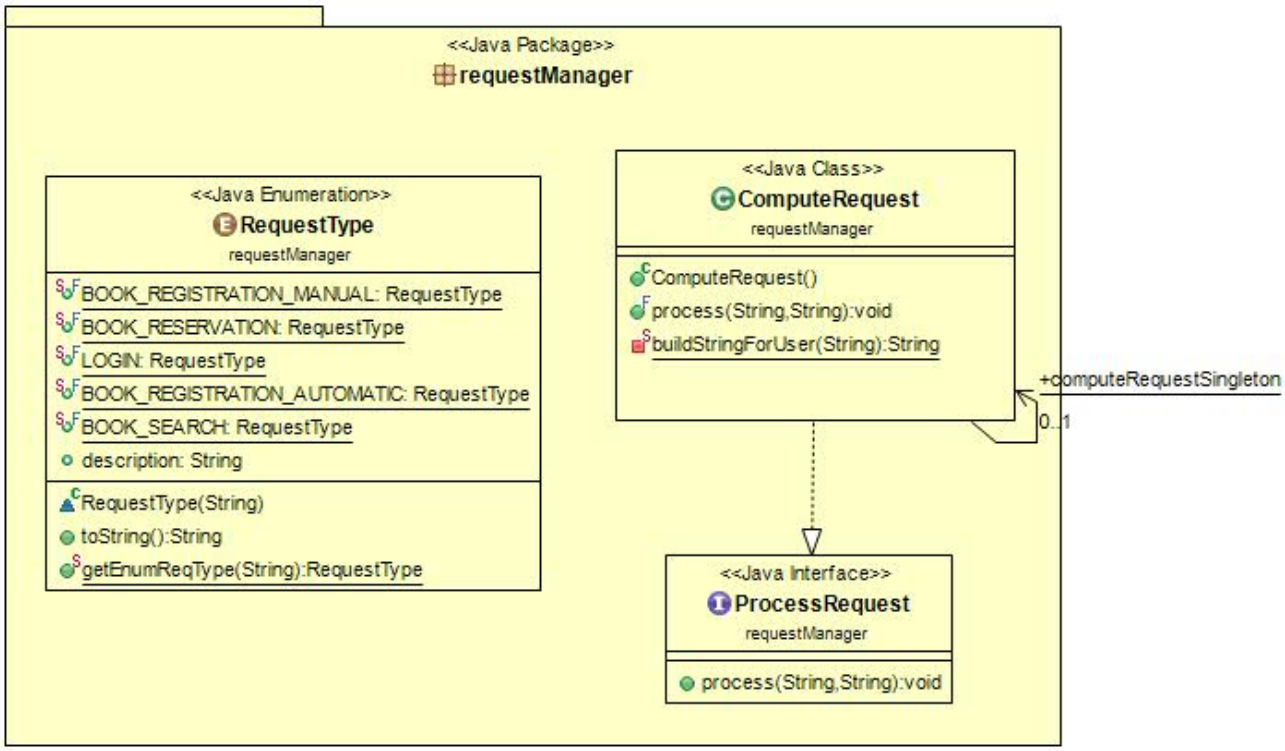

Figure 9.1: Diagramma della classe ComputeRequest

Si può notare infatti come alla ***RequestType*** si aggiungano 3 casi in più, relativi proprio ai nuovi componenti sviluppati in questa seconda iterazione. Allo stesso modo anche il codice si completa così

```
226 case BOOK_RESERVATION:
227   boolean rs = book.reserve(username);
228   Communication.getInstance().send(username, "requestType:1;result:" + (rs?1:0));
229   break;
230 case LOGIN:
231   Communication.getInstance().send(username, "requestType:2;result:" + "Success" + ";
        " + buildStringForUser(username));
232   break;
233 case BOOK_REGISTRATION_AUTOMATIC:
234   result = b.insert();
235   Communication.getInstance().send(username, "requestType:4;result:" + (result?1:0) +
        ";BCID:" + b.getBCID());
236   break;
```

Listing 9.1: Tipologia di richieste aggiuntive gestite durante la seconda iterazione

Questo si va ad aggiungere a quanto sviluppato nella prima iterazione: a questo punto possiamo quindi gestire le richieste del client compilate nel formato descritto relative ai componenti disponibili nella prima e seconda iterazione.

10

Parte algoritmica: *reservation handler*

La gestione delle prenotazioni dei libri è una delle parti innovative introdotte dalla start up: questo servizio mira a sfruttare la flessibilità della community di sharing, basata sull'idea dell'open-source, cercando comunque di offrire un servizio mirato ed attento alle necessità del lettore.

Ogni utente, purchè sia registrato all'interno del servizio di Book-sharing, può prenotare un determinato libro che si trova nello stato "Under reading", ovvero si trova in mano ad un altro utente, il quale lo sta leggendo.

Andiamo ad evidenziare gli attori coinvolti in questa operazione di prenotazione:

- **Lettore [L]:** esso rappresenta l'utente, registrato nella community, che possiede il libro oggetto della prenotazione. Indichiamo con:
  - r_L**:** raggio d'azione del lettore;
  - z_0**:** zona di residenza (espressa come coordinate puntuali).
- **Prenotanti** [P_i **con** $i = 1, ..., N$]**:** rappresentano l'insieme degli N utenti, tutti interessati ad uno specifico libro in possesso dell'utente **L**.

  Oltre a questa informazione, ogni utente avrà fornito, al momento della registrazione, le seguenti informazioni:
  - r_i^P **con** $i = 1, ..., N$**:** raggio d'azione del lettore;
  - z_i^P **con** $i = 1, ..., N$**:** zona di residenza.

  L'algoritmo può dunque essere scomposto in due macro-blocchi:
  - **Step 0:** questa fase viene richiamata nel momento in cui il sistema inizia ad analizzare tutti gli utenti che hanno effettuato una prenotazione per un determinato libro che si trova nello stato di *"Under reading"*.

    Tutti gli N prenotanti P_i vengono ordinati in base alla distanza dal lettore L, indipendentemente da quello che è l'ordine temporale con cui è stata effettuata la prenotazione: la quantità di cui si terrà (come si può vedere nella figura 10.1) conto sarà quindi la distanza

    $$|z_i^p - z_0| \tag{10.1}$$

    Questo ordinamento corrisponde quindi sostanzialmente a creare una *priority queue* in cui si va ad assegnare una maggiore priorità all'utente la cui zona di residenza è più vicina a quella del lettore in possesso del libro richiesto.
  - **Step 1:** in questo macro-blocco andiamo effettivamente ad applicare l'algoritmo *smart* per poter soddisfare, nella maniera migliore, le esigenze di ogni utente della community.

    L'idea di base è che, se utente lettore e utente prenotante hanno possibilità di incontrarsi, ovvero se i loro raggi d'azione si sovrappongono, essi potranno accordarsi direttamente

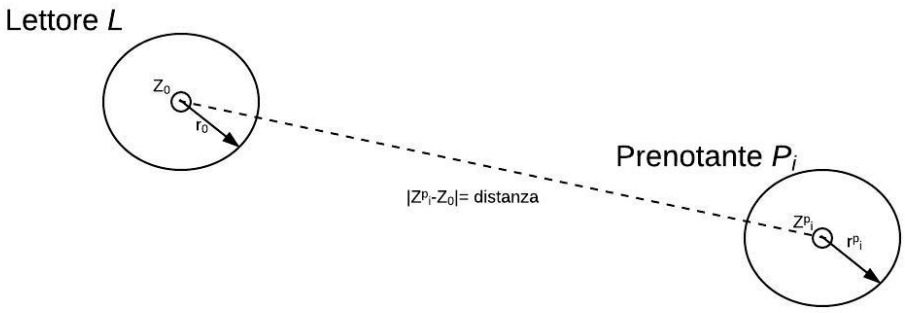

Figure 10.1: Distanza tra lettore e prenotante i-esimo

sul luogo dello scambio, rendendo *"safety"* il passaggio del libro: questo scambio avverrà ovviamente in una zona all'interno dell'intersezione dei raggi d'azione, come mostrato in figura 10.2.

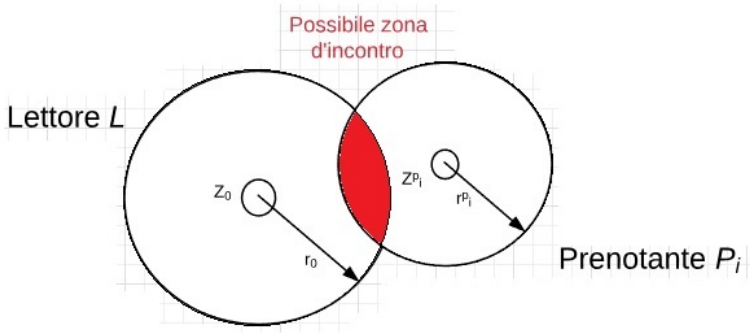

Figure 10.2: Zona d'incontro tra lettore e prenotante

Nel caso in cui invece, i due utenti interessati non abbiano la possibilità di trovare un luogo comune in cui potersi scambiare il libro fisicamente si avrà che, la rete di users appartenenti alla community farà da tramite, per portare il libro *"coast-to-coast"*.
Quindi, tramite un semplice pseudo-codice, possiamo descrivere il nostro algoritmo come

Algorithm 1: Algoritmo di gestione della prenotazione

```
Data: Informazioni dell'utente lettore e di quello prenotante
Result: Percorso ottimo dal lettore al prenotante
Step 0 (inizializzazione);
if Distanza <= 0 then
    Trova un punto d'incontro nell'unione delle delle area;
    Notifica gli utenti di dove potersi scambiare direttamente il libro;
else
    Crea la rete di utenti che faranno da tramite tra lettore e prenotante;
    Ricercare il cammino ottimo (il libro si muoverà hand-to-hand);
end
```

Nello specifico il calcolo della distanza avverrà tramite la funzionalità *checkOverlap(Lettore, Prenotante)*, la quale andrà a verificare che:

$$|z_i^p - z_0| - r_0 - r_i <= 0 \tag{10.2}$$

ovvero che i raggi d'azione si sovrappongano o meno.

Nel caso in cui i due utenti non abbiano possibili punti d'incontro (distanza >= 0), dobbiamo selezionare l'insieme di utenti tramite i quali il libro in questione potrà spostarsi: l'idea base è quindi quella di costruirsi un'area circolare di centro pari alla metà della congiungente del punto z_0 (zona di residenza lettore) e z_i^p (zona di residenza prenotante). Verranno poi selezionati tutti gli utenti che si trovano all'interno di questa circoscrizione.

In passi sequenziali, possiamo scrivere:

Algorithm 2: Creazione del percorso tra lettore e prenotante

Data: Zona di residenza e raggio d'azione di utente lettore e prenotante
Result: Elenco di utenti attraverso cui il libro dovrà spostarsi *hand-to-hand*
Il raggio della circoscrizione di utenti coinvolti sarà pari alla distanza

$$\bar{Z} = \frac{1}{2}|z_i^p - z_0|$$

for *ogni utente* z_i^U *che si trova all'interno della community* **do**
  if $(Distanza(z_0, z_i^U)) <= \bar{Z}$ *oppure* $(Distanza(z_i^P, z_i^U)) <= \bar{Z}$ **then**
    Seleziono l'utente z_i^U e lo inserisco nella lista (*HandToHandUsers*) dei possibili utenti che potrebbero partecipare attivamente al prestito;
  end
end
Creiamo il collegamento tra gli users della lista *HandToHandUsers* il cui raggio d'azione si sovrappone;

Quindi, alla fine dello step 1, avremo individuato tutti gli utenti i quali possono partecipare attivamente alla realizzazione di un prestito: il risultato ottenuto sarà quindi come quello in figura 10.3.

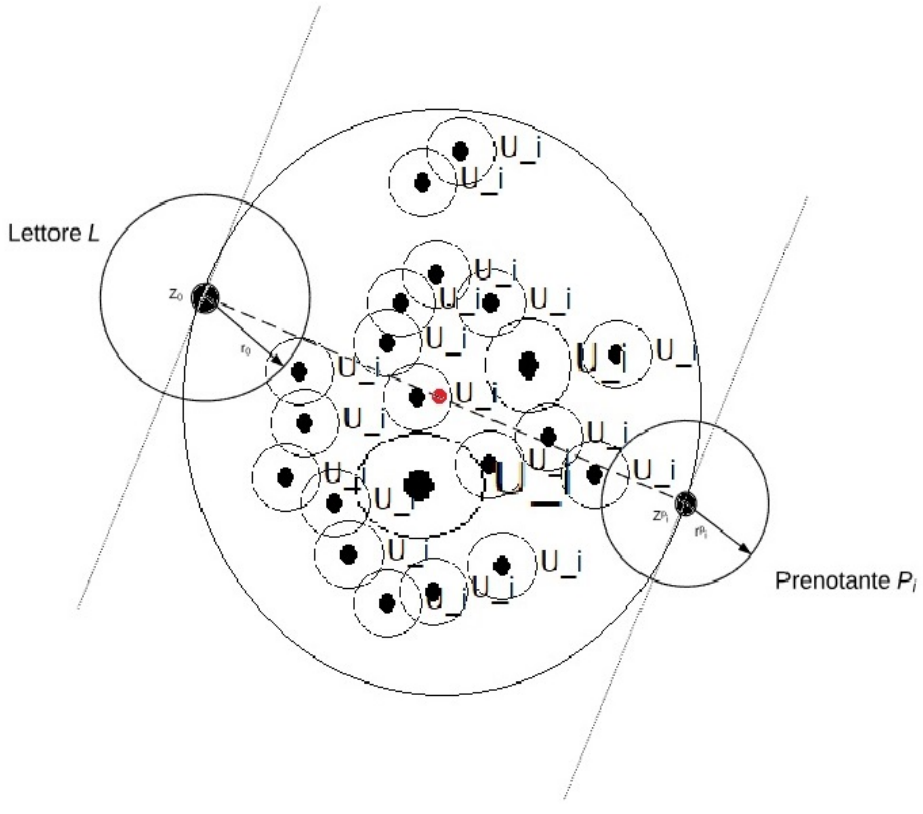

Figure 10.3: Rete di utenti che potrebbero essere attivi nel prestito

Un ulteriore miglioramento che è stato introdotto è rappresentato dal comportamento greedy dell'algoritmo *decisionale*: come è facilmente intuibile, in questo contesto, l'obbiettivo base è quello di andare a minimizzare il numero di km percorsi dal libro, per giungere al lettore prenotante.

Il concetto è quindi quello di minimizzare il numero di utenti che prenderanno parte attivamente al prestito, realizzando un *passamano* tra uno e l'altro.

Un approccio classico per minimizzare il percorso seguito dal libro durante il prestito è quello di andare a scegliere l'utente successivo a cui far arrivare il libro come l'utente la cui distanza è minima tra tutti quelli possibili.

Per rendere però più performante possiamo andare ad applicare un approccio *greedy*, il quale consente di aumentare l'ottimalità dell'algoritmo stesso: nello specifico il concetto seguito nell'implementazione è stato quello di andare ad optare tra due differenti scelte decisionali, ovvero:

- *Probabilità* ϵ: andiamo a scegliere l'utente più vicino, tra tutti quelli selezionabili;
- *Probabilità* $1-\epsilon$: tra gli utenti che si possono selezionare, si va a scegliere, con questa probabilità, l'utente con raggio d'azione maggiore

Grazie a questa duplice possibile scelta, siamo in grado di rendere l'algoritmo più ottimale, ovvero in grado di trovare, nella maggior parte dei casi, un percorso ottimo; in particolare, minore è il valore di ϵ che andiamo a scegliere, più *forte* sarà l'algoritmo, ovvero troverà sempre un percorso ma senza garanzia che sia quello ottimale. Maggiore invece è il valore di ϵ scelto, più bassa sarà la probabilità che troverà un percorso però, il percorso trovato, sarà uno dei più corti.

11

Test ed analisi dei componenti implementati

11.1 Lato Android

Per quanto rigurda la fase di testing di quanto implementato per il client Android sono stati sfruttati principalmente due tool:

- **Junit** per l'analisi dinamica del codice, con lo scopo quindi di verificare che il codice scritto funzioni in maniera coerente a quanto richiiesto e che vengano seguite le richieste descritte nei casi d'uso.
- **Espresso Junit** per l'analisi dell'interfaccia grafica costruita per la gestione delle varie richieste. Anche in questo caso si è fatto riferimento a quanto descritto dai casi d'uso riguardo alle view e al loro formato per osservare la correttezza di quanto implementato.

11.1.1 Espresso Junit

Come detto, **Espresso Junit** è un tool attraverso il quale è possibile verificare le interazioni di un applicativo Android con l'utente. In questo caso quindi gli *assert* verificano, ad esempio, che il fragment aperto sia quello corretto e che i campi dei vari componenti inseriti abbiano il valore desiderato.

Nell'ambito di questa iterazione, le componenti che sono state testate con questo applicativo sono quelle di:

1. Login
2. Registrazione manuale di un libro
3. Ricerca di un libro

Consideriamo, come esempio per la descrizione, il test relativo alla ricerca di un libro all'interno della rete di Book Crossing. Il test si compone nel seguente modo:

```
237 @Test
238 public void searchBookTestOK(){
239   Globals.isLoggedIn = true;
240   Globals.usernameLoggedIn = "A";
241   onView(withId(R.id.navigation)).check(matches(isDisplayed()));
242   onView(withId(R.id.book_search)).perform(click());
243   onView(withId(R.id.book_search)).check(matches(isDisplayed()));
244
245   ViewInteraction checkTitle = onView(withId(R.id.title_search));
246   ViewInteraction checkAuthor = onView(withId(R.id.author_search));
247   ViewInteraction checkBtnSearch = onView(withId(R.id.search_button));
248
249   checkTitle.check(matches(isDisplayed()));
250   checkAuthor.check(matches(isDisplayed()));
```

```
checkAuthor.check(matches(isDisplayed()));

checkTitle.perform(setTextInTextView(""));
checkAuthor.perform(setTextInTextView(""));

checkTitle.perform(setTextInTextView("questa storia"));
checkBtnSearch.perform(click());

onView(withId(R.id.books_found)).check(matches(isDisplayed()));
onData(anything()).inAdapterView(withId(R.id.books_found)).
atPosition(0).perform(click());
onView(withText(containsString("BOOKABLE"))).
inRoot(withDecorView(not(mainActivityActivityTestRule.getActivity().getWindow().
getDecorView()))).check(matches(isDisplayed()));
```

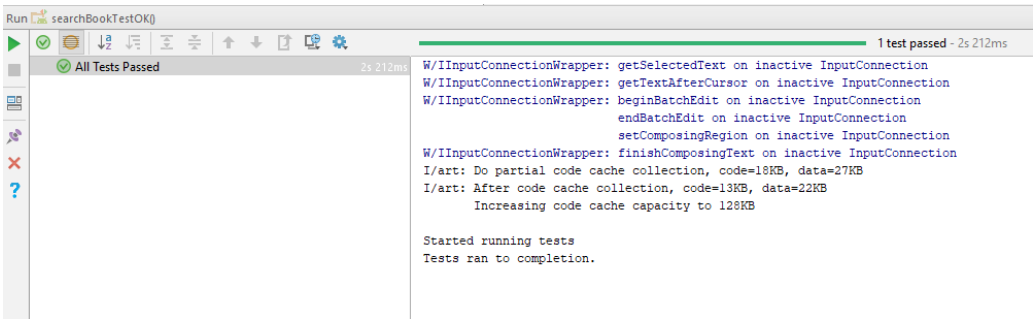

Figure 11.1: Espresso test

Questo caso di test verifica che la ricerca di un libro venga svolta in maniera corretta, garantendo la visualizzazione dei corretti *fragment* e dei *Toast* formattati in maniera corretta. L'esito di questo caso di test eseguito attraverso il tool **Espresso Junit** fornito da Android Studio può essere riassunto dalla figura 11.1. La medesima procedura è stata implementata per le altre componenti descritte precedentemente; il fatto di utilizzare questo tool consente di svolgere anche un'analisi dinamica del codice, dal momento che se ci fosse un errore, la successione grafica subirebbe un crash, mostrando un segnale di errore e facendo fallire il caso di test previsto.

11.2 Lato Server

11.2.1 Analisi Dinamica

JUNIT Test

In questa sezione viene presentato un esempio di analisi dinamica del codice per quanto riguarda il lato server. Come esempio da riportare nella documentazione si è scelto di valutare il caso di test relativo alla fase di login, sia nel caso in cui l'esito sia positivo sia in caso in cui l'esito sia negativo.

```
    @Test
    public void logintTest() {

    Profile u = new Profile();
    u.setUsername("A");
    u.setPassword("6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b")
    ;
    assertTrue(LoginStatus.SUCCESS == u.login());

    u.setPassword("1");
    assertFalse(LoginStatus.SUCCESS == ProfileData.getInstance().login(u));
    assertTrue(LoginStatus.WRONG_PWD == ProfileData.getInstance().login(u));

    u.setUsername("paperino");
    assertTrue(LoginStatus.WRONG_USERNAME == ProfileData.getInstance().login(u));
```

```
279     }
280
281     @Test
282     public void existLogin() {
283     Profile u = new Profile();
284     u.setUsername("A");
285
286     assertTrue(Profile.existUser(u.getUsername()));
287
288     u.setUsername("ZX");
289     assertFalse(Profile.existUser(u.getUsername()));
290     }
291
```

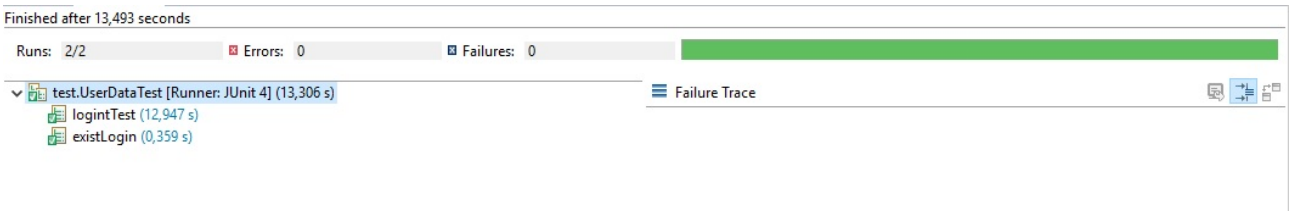

Figure 11.2: Junit test

L'esito è indicato dalla figura 11.2, la quale evidenzia il fatto che inserendo username e password corretti, il login ha un esito successivo, al contrario si verifica che con delle credenziali sbagliate si verifica un errore.

11.2.2 Analisi Statica

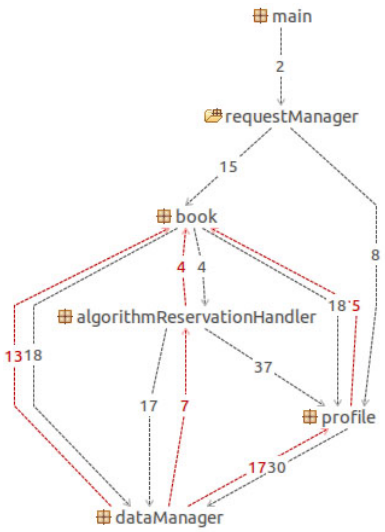

Figure 11.3: Analisi statica del codice

TODO:commentare immagine e controllare che corrisponda a quanto pianificato precedentemente
Nel seguente piano sono rappresentate le metriche di:

- Abstractness: misura quanto facilmente il sistema può essere espanso;
- Instability: tentativo di misurare la facilità di cambiamento.

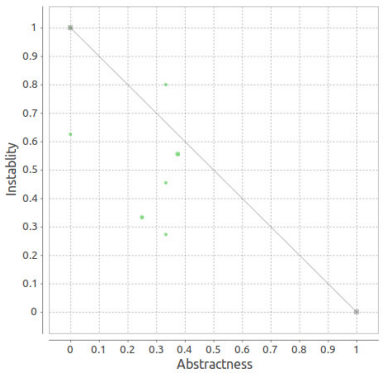

Figure 11.4: analisi statica grafico

I packages vicino al punto (0,0) sono rigidi mentre i packages vicino a (1,1) sono totalmente astratti e quindi inutili, idealmente vorremmo trovarci sulla linea rappresentata.