La Corriera
correzione esercitazione

Laura Semini, Ingegneria del software
Corso di Laurea in Informatica, Università di Pisa

Legenda

Nessun commento = 

Commento = 

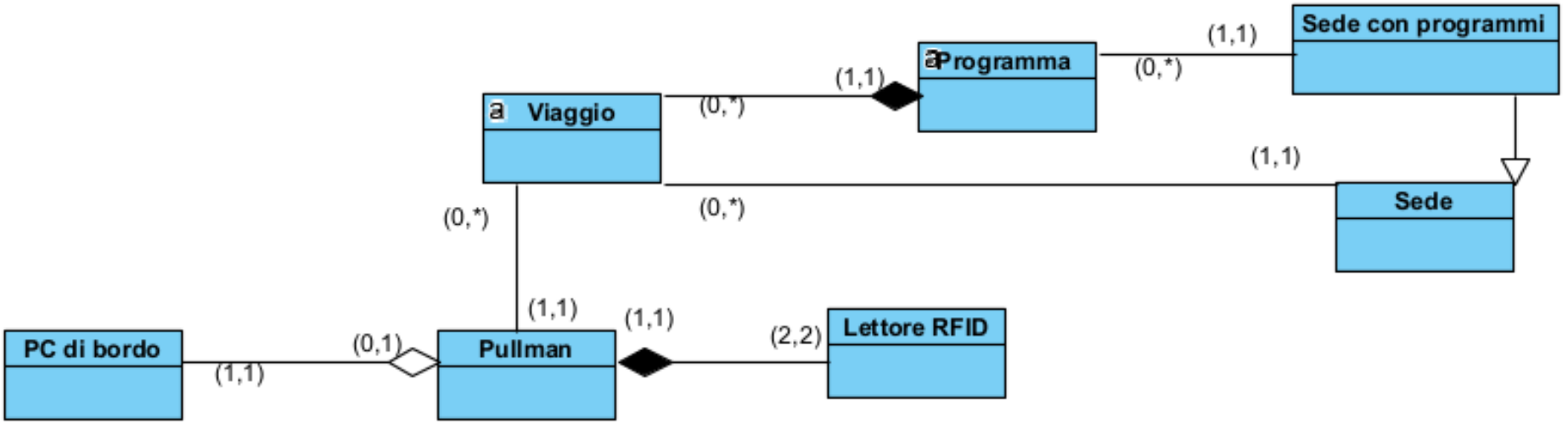
Commento !!!!! = 

Compito standard: 5 esercizi, max voto 5 a esercizio, poi normalizzo la somma a 33 (somma : voto = 25 : 33)

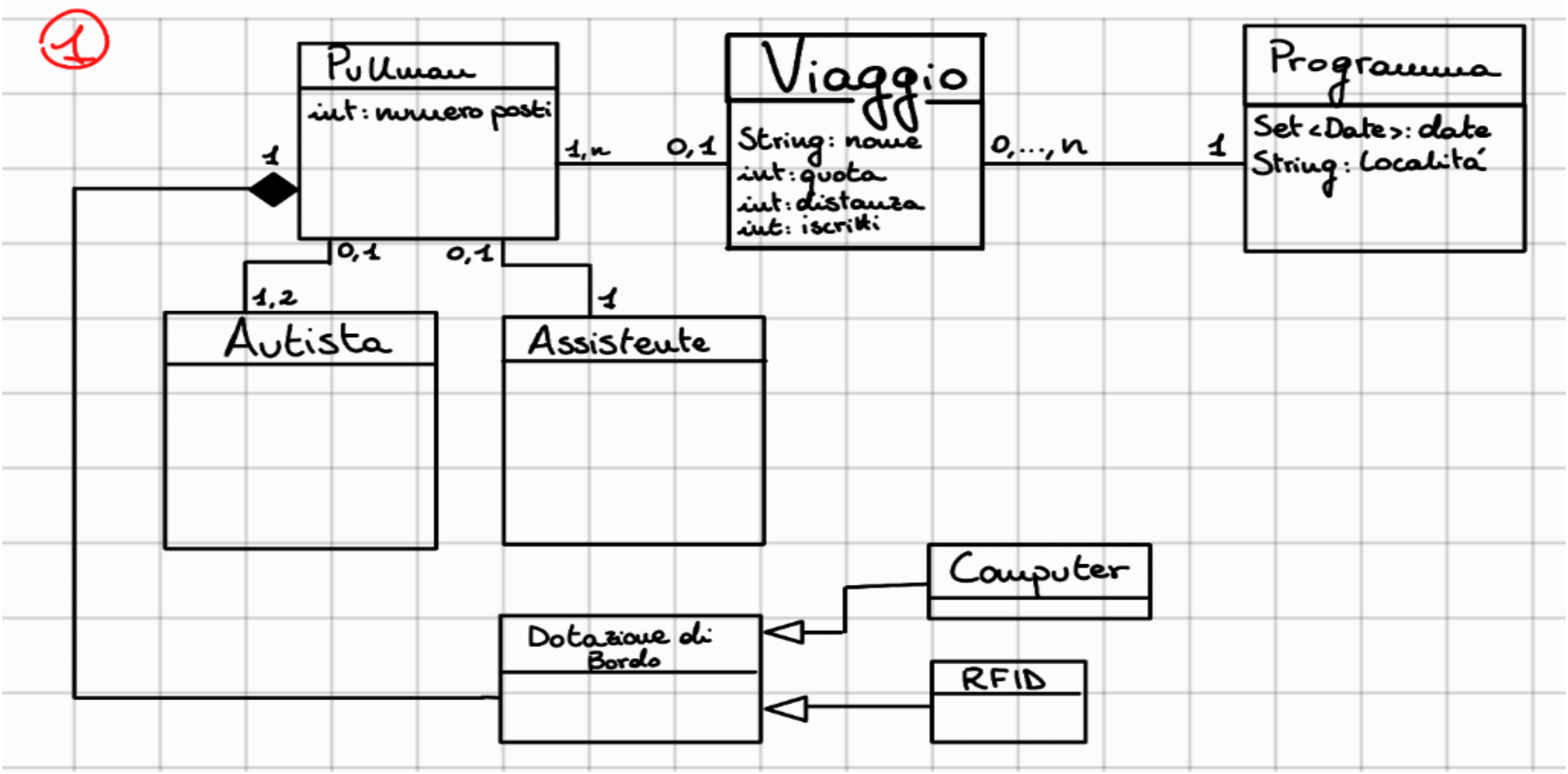
Un errore toglie min 0.5 , max 2

Esercizio 1

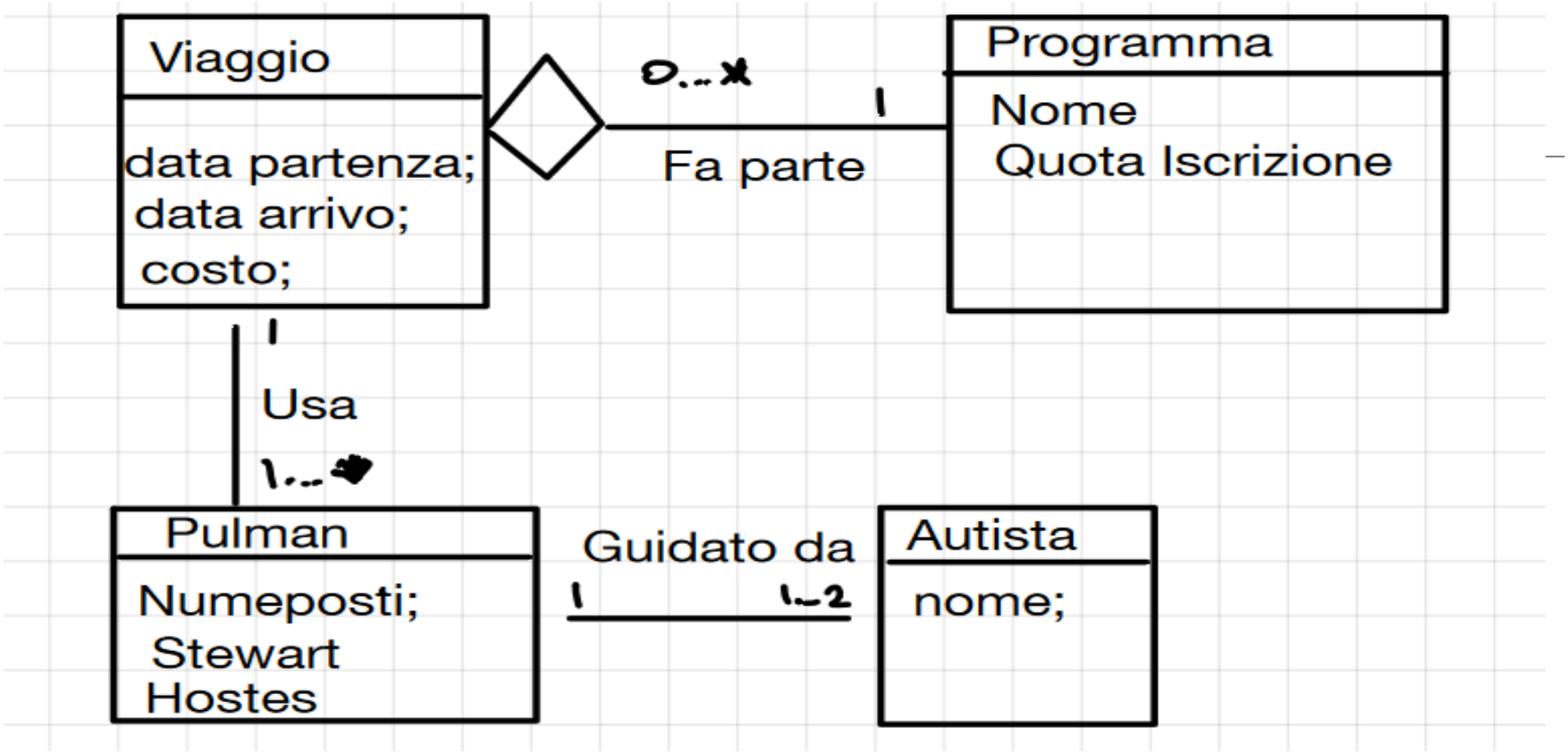
Analisi del dominio. Dare un diagramma delle classi che modelli la parte di dominio descritta nel terzo e nel quarto paragrafo dell'enunciato (*Ogni viaggio è relativo a un solo programma, che è contraddistinto da un nome e ha associata una quota di iscrizione. Un programma di viaggio può essere attuato in più date: i viaggi relativi differiscono tra loro, a parte per le date di partenza/ritorno, per il costo (soggetto a variazioni stagionali, distinto tra adulti e bambini - considerati tali se di età inferiore a 12 anni -) e per il numero di posti sul pullman.*
A bordo di ogni pullman ci sono un autista (due se sono previste tappe superiori ai 500 km) e uno steward o una hostess.) e che includa anche le dotazioni hardware di bordo.



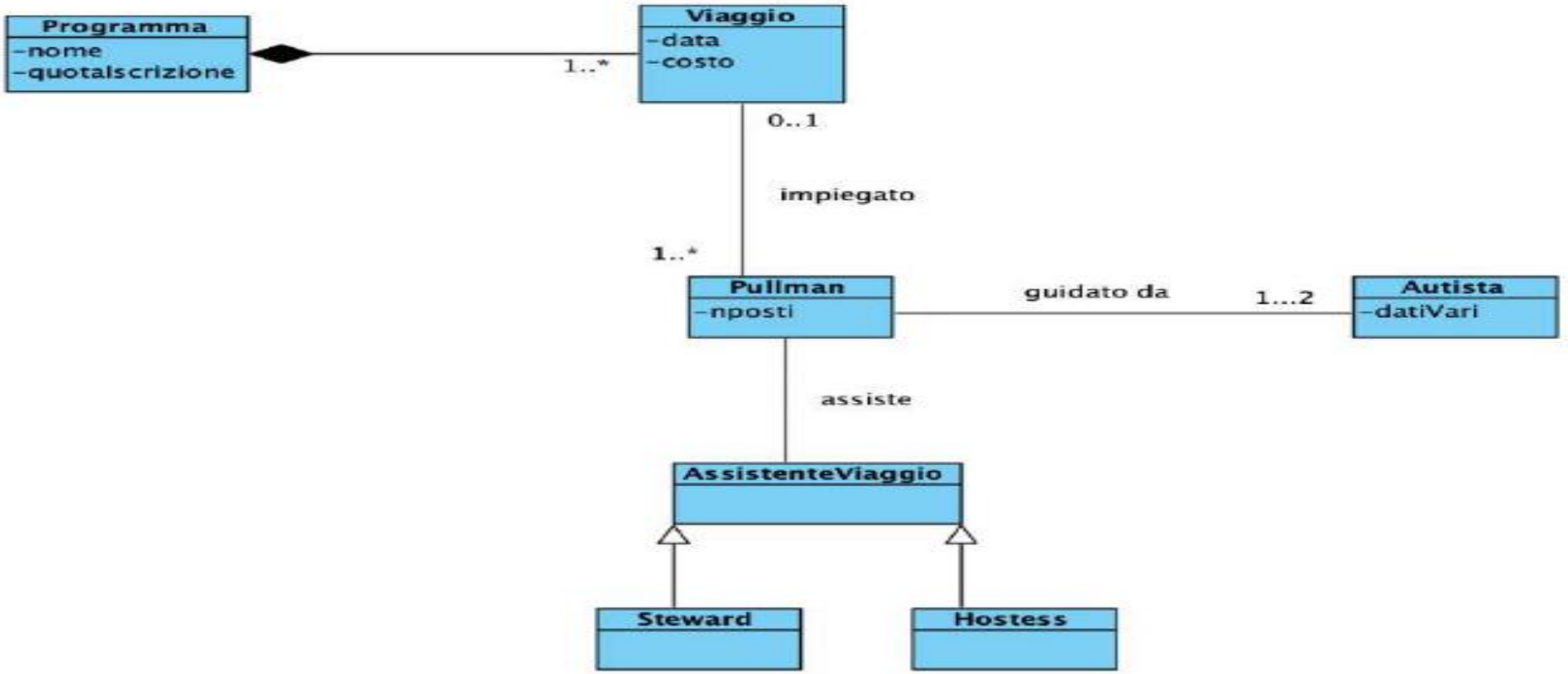
- Mancano i nomi delle associazioni
- Molteplicità vanno senza parentesi; 1,1 == 1 e si può anche omettere; 0,* ==*
- Mancano: nome, quota, data, posti pullman, autista (due se sono previste tappe superiori ai 500 km) , steward o hostess.
- I lettori comunicano con il PC (ma non è errore errore non metterlo)



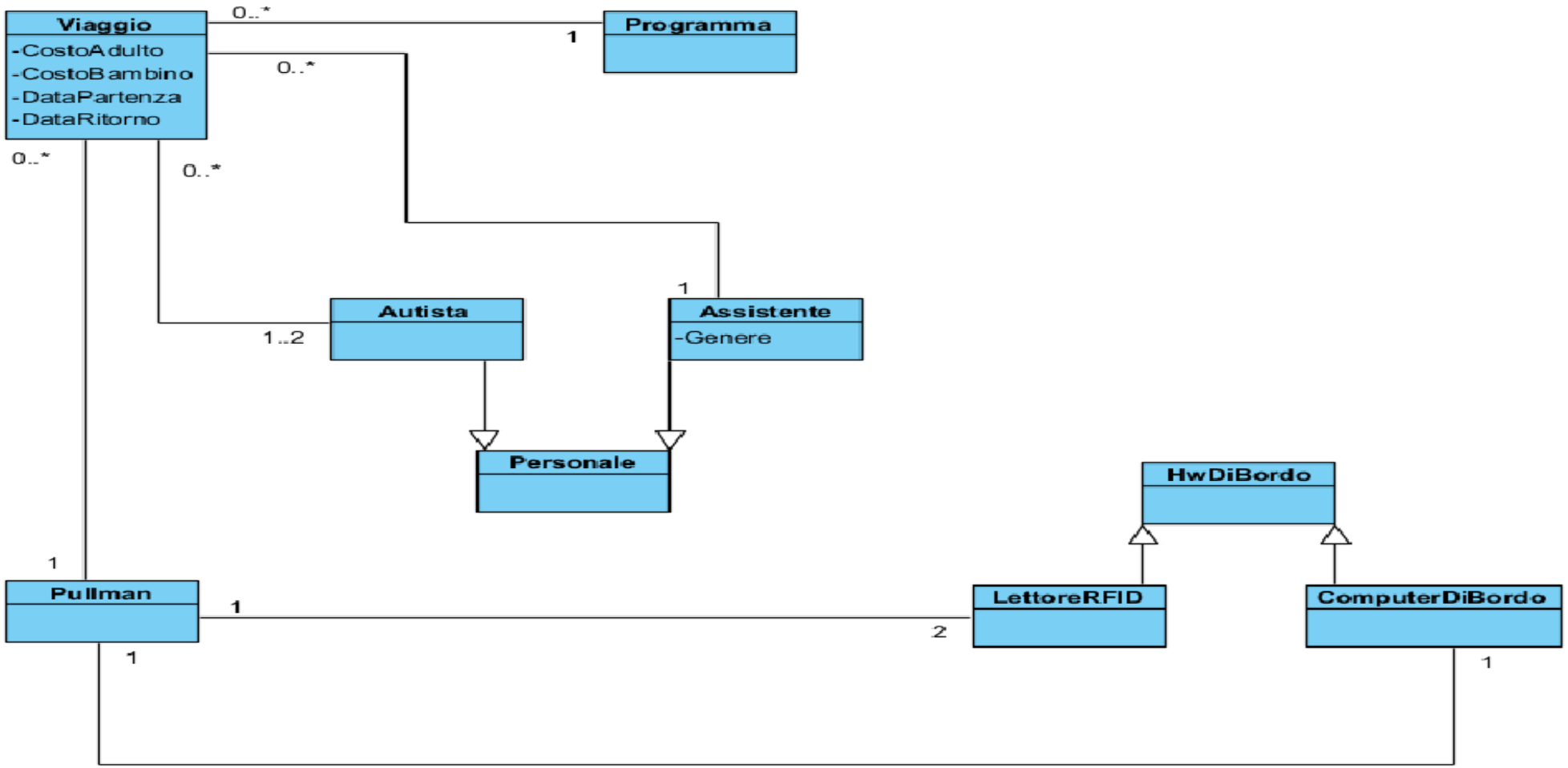
- Mancano i nomi delle associazioni
- Manca molteplicità dotazione di bordo (problema legato uso ereditarietà)
- Le date sono del viaggio. Distanza-> meglio booleano maggiore 500km; Iscritti????? località????



- Aggregazione al contrario: il programma è aggregazione di viaggi
- Il nome dell'autista è inutile
- Steward e Hostess come attributi distinti: NO
- Manca hw di bordo



- Manca hw di bordo
- Dati vari: ambiguo e inutile



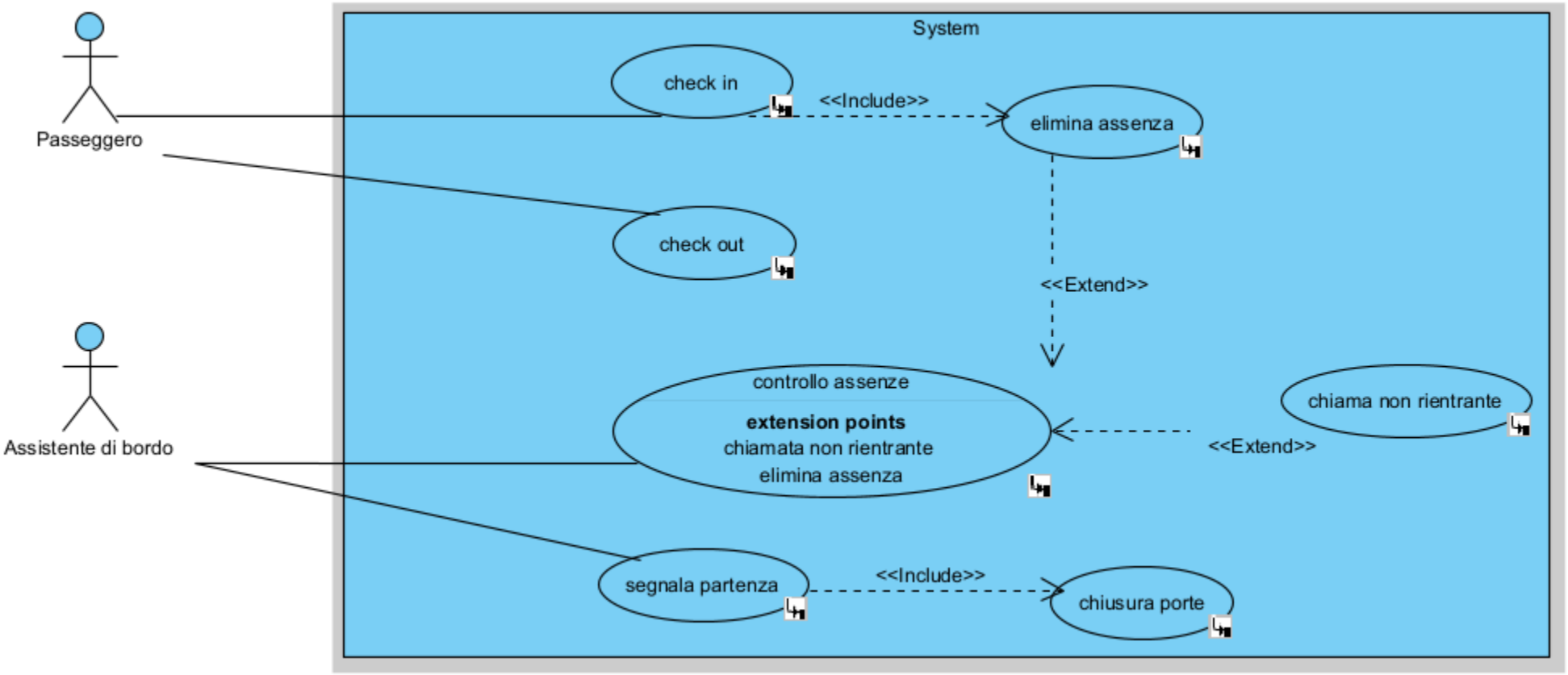
- Mancano nomi associazioni e alcune sono aggregazioni

Riassunto

- Nomi associazioni: metterli. Aggregazioni, composizioni non ce l'hanno (sottointeso: è formato da)
- Non dimenticate nulla
- Non inventate nulla

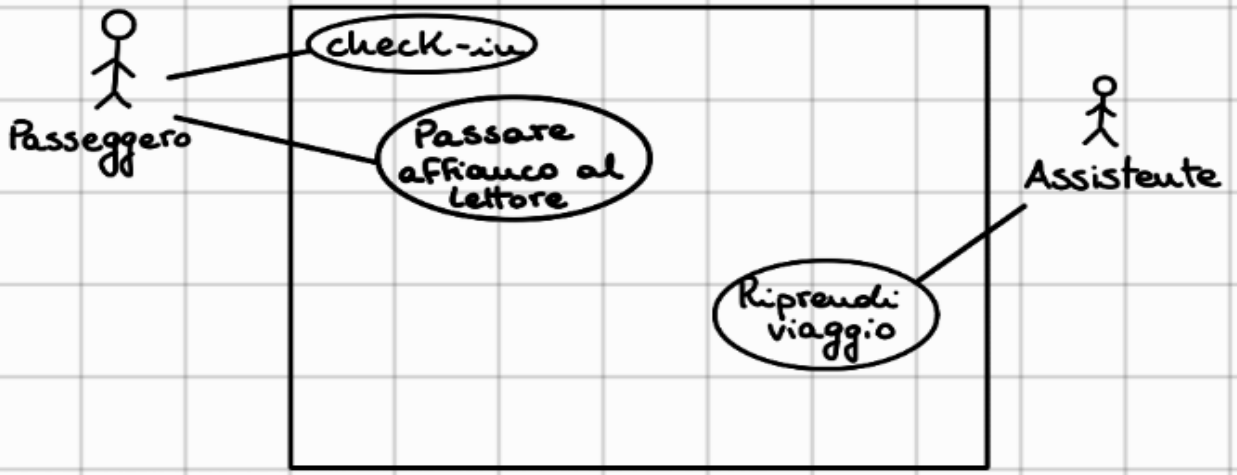
Esercizio 2

Requisiti. Dare il diagramma dei casi d'uso che riguardano il controllo delle presenze a bordo. Si noti che è necessaria un'attività di Check-in per inizializzare lo stato del sistema. Per tutti i casi d'uso dare pre- e post-condizione



- Mancano pre-post (?)
- Check-out??
- Il sistema non supporta la chiamata al non rientrante (ok, era ambiguo)
- Non capisco elimina assenza vs check in
- Scelta opinabile che sia il passeggero a fare il check in

2



Check-in: precondizione: avere il biglietto

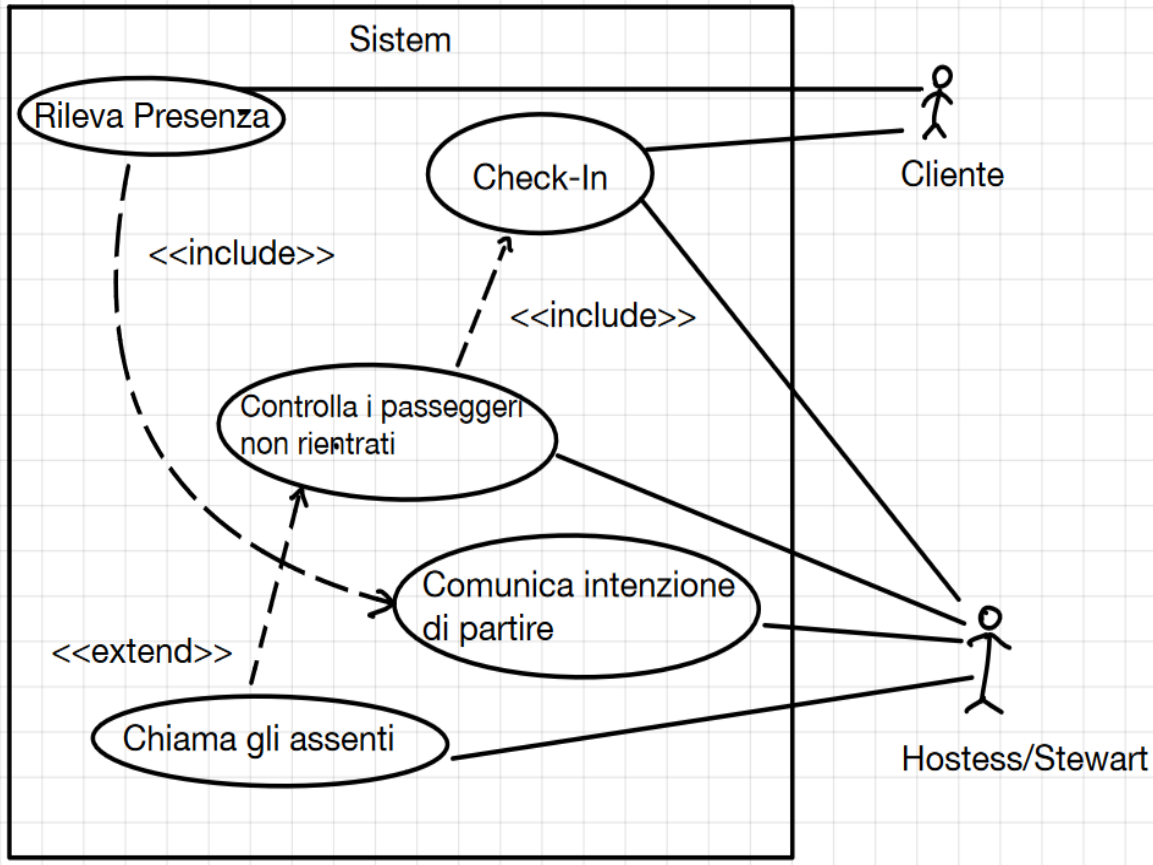
postcondizione: ottenere la tessera

Riprendi viaggio: precondizione: essere in sosta

postcondizione: verifica passeggeri non rientrati

Passare [...]: precondizione: avere la tessera

postcondizione: passeggero conteggiato



- Check-In
  Attore Primario: Cliente
  Attore Secondario: Hostess/Stewart
  Pre: Viaggio prenotato dal cliente
  Post: Tessera RFID consegnata, cliente registrato nel sistema di presenze

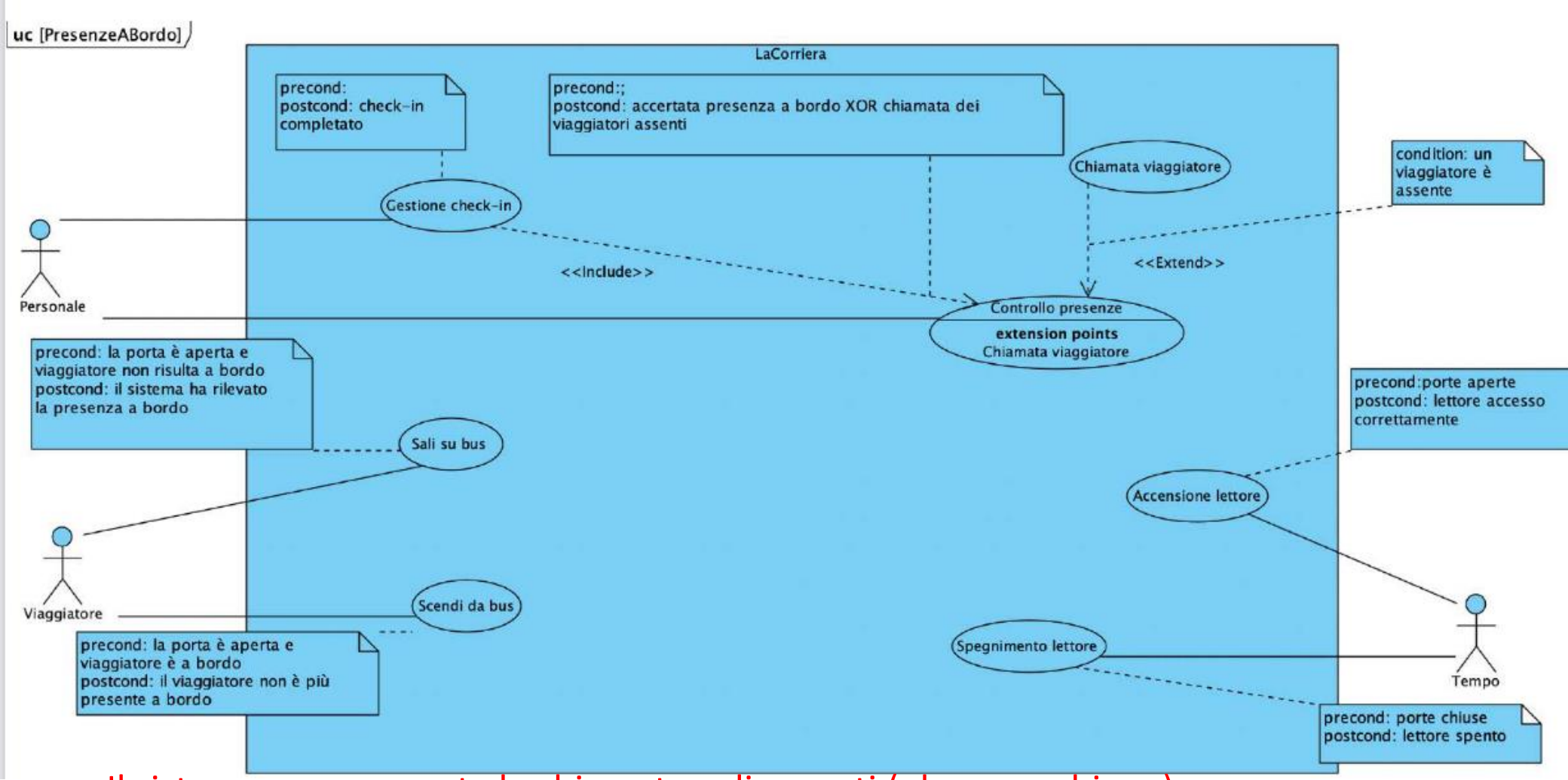
- Comunica intenzione di partire
  Pre: Pulman pronto
  Post: Porte aperte, sistema di verifica presenze pronto

- Controlla i passeggeri non rientrati
  Pre: Porte aperte, hostess pronta a controllare gli assenti
  Post: Lista assenti sul PC di bordo.

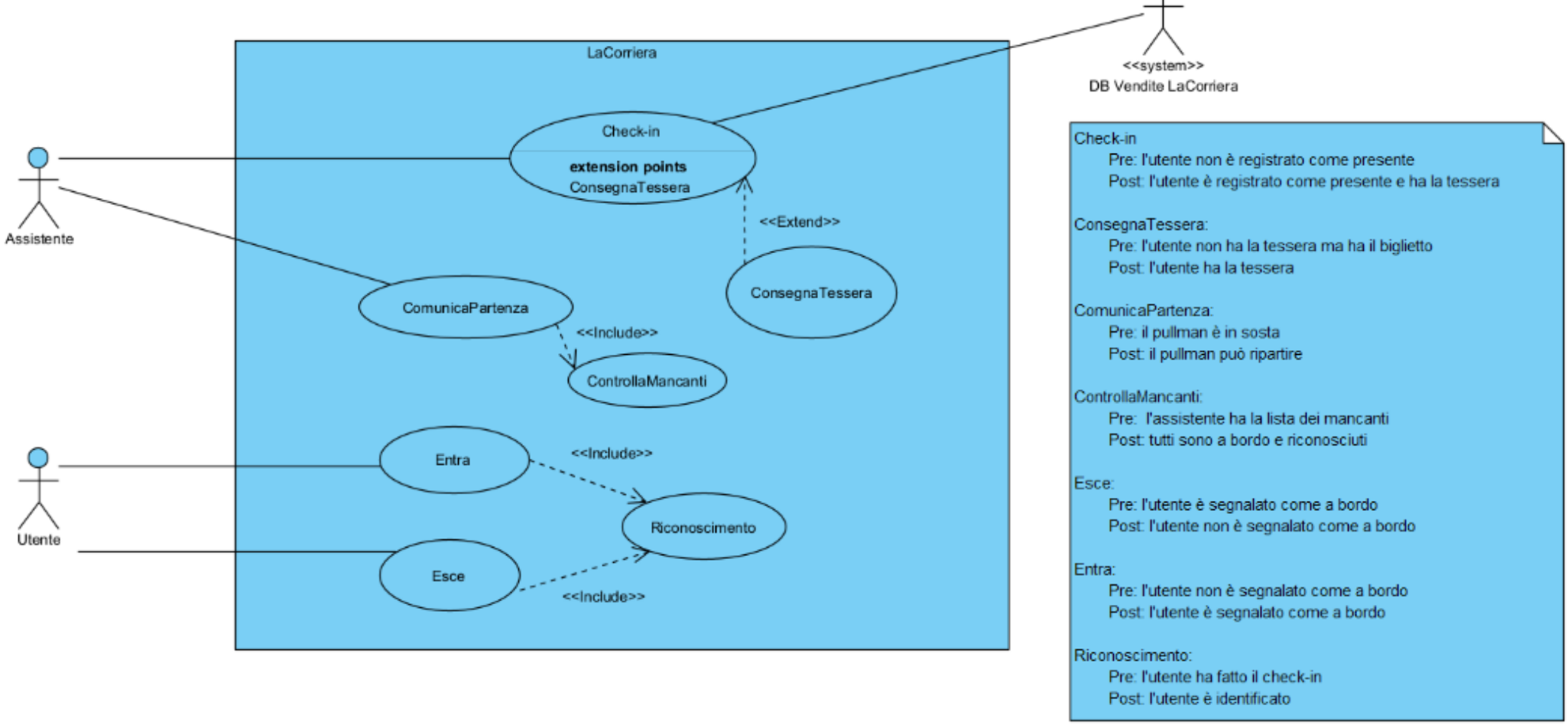
- Chiama glia assenti
  Pre: Lista assenti sul PC
  Post: Clienti assenti chiamati

- Ricevi Presenza
  Pre: Sistema acceso, passeggeri con RFID
  Post: Sistema Aggirnato

- Il sistema non supporta la chiamata agli assenti (ok, era ambiguo)
- Include verso sbagliato
- Scelta opinabile che sia il passeggero a fare il check in
- Precondizioni ambigue o inutili imprecise: Pullman pronto, lista assenti sul PC



- Il sistema non supporta la chiamata agli assenti (ok, era ambiguo)
- Include ???
- Opinabile che sia il tempo ad accendere/spengere il lettore. Meglio le porte
- Postcondizione di "Scendi…": il sistema ha rilevato…, quella attuale inutile



- DB vendite fa parte del sistema!!!!
- Meglio assegna tessera che "consegna", la parte di consegna fisica non è supportata dal sistema
- Postcondizione di "Scendi…": il sistema ha rilevato…, quella attuale inutile

Riassunto

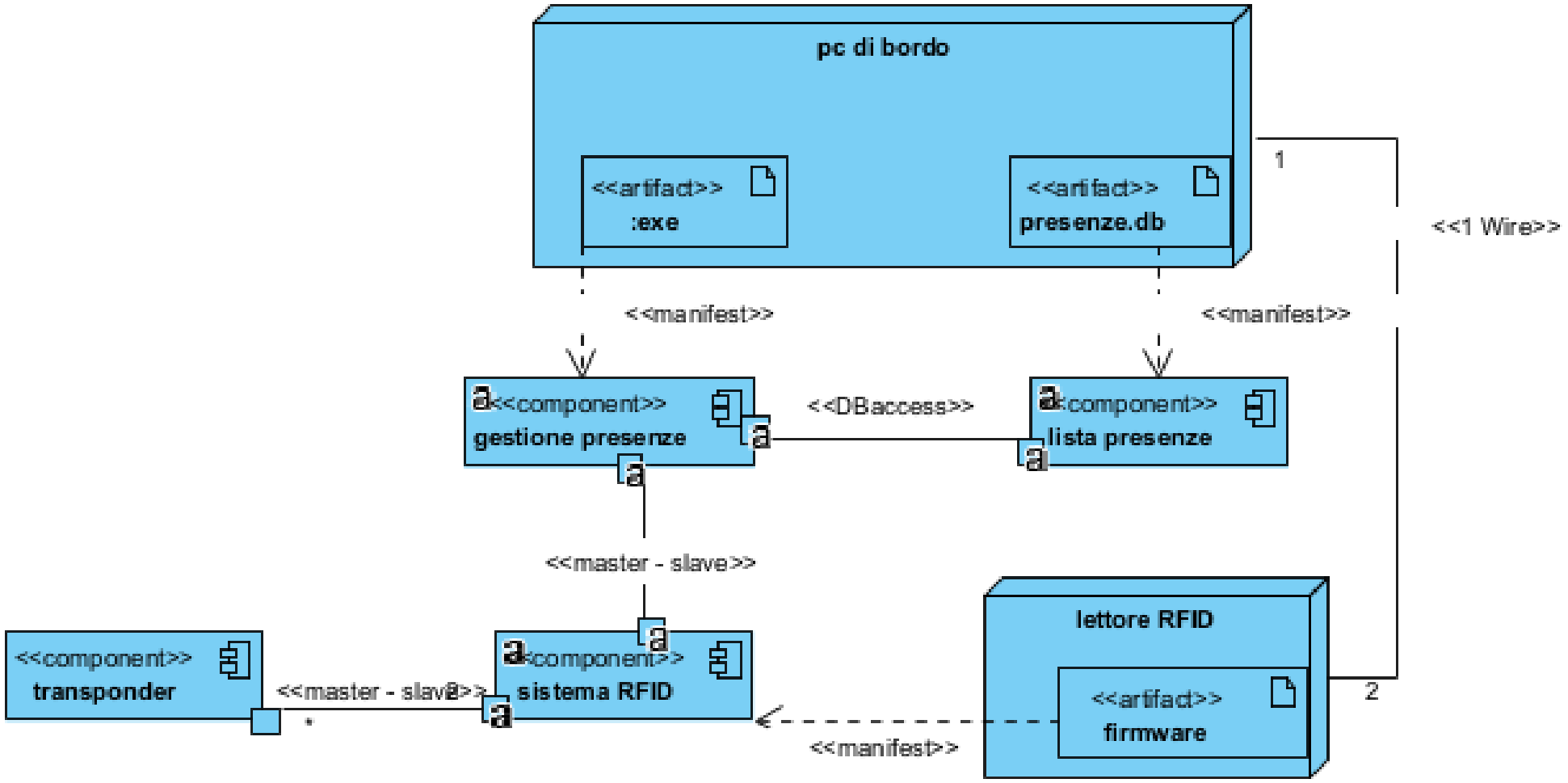
- Non dimenticate nulla
- Non inventate nulla
- Le precondizioni possono essere *True*
- Ricordarsi cosa è sistema e cosa no
- Attenzione a quali task sono effettivamente supportati dal sistema
- Pre e post: pensare anche (soprattutto) allo stato del sistema

Esercizio 3

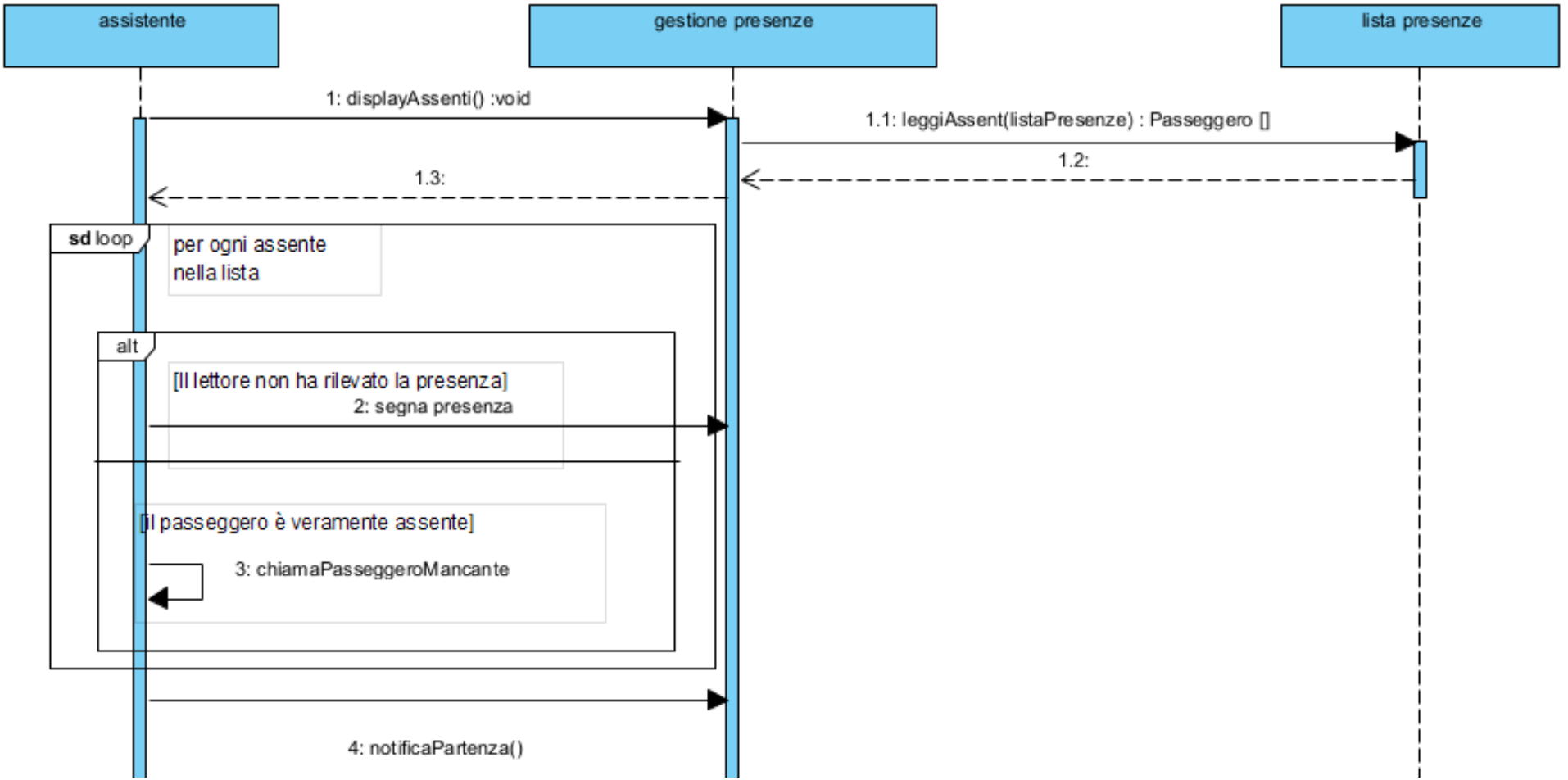
Architettura. Dare la vista ibrida C&C più dislocazione delle componenti necessarie per implementare il controllo delle presenze a bordo.

Esercizio 4

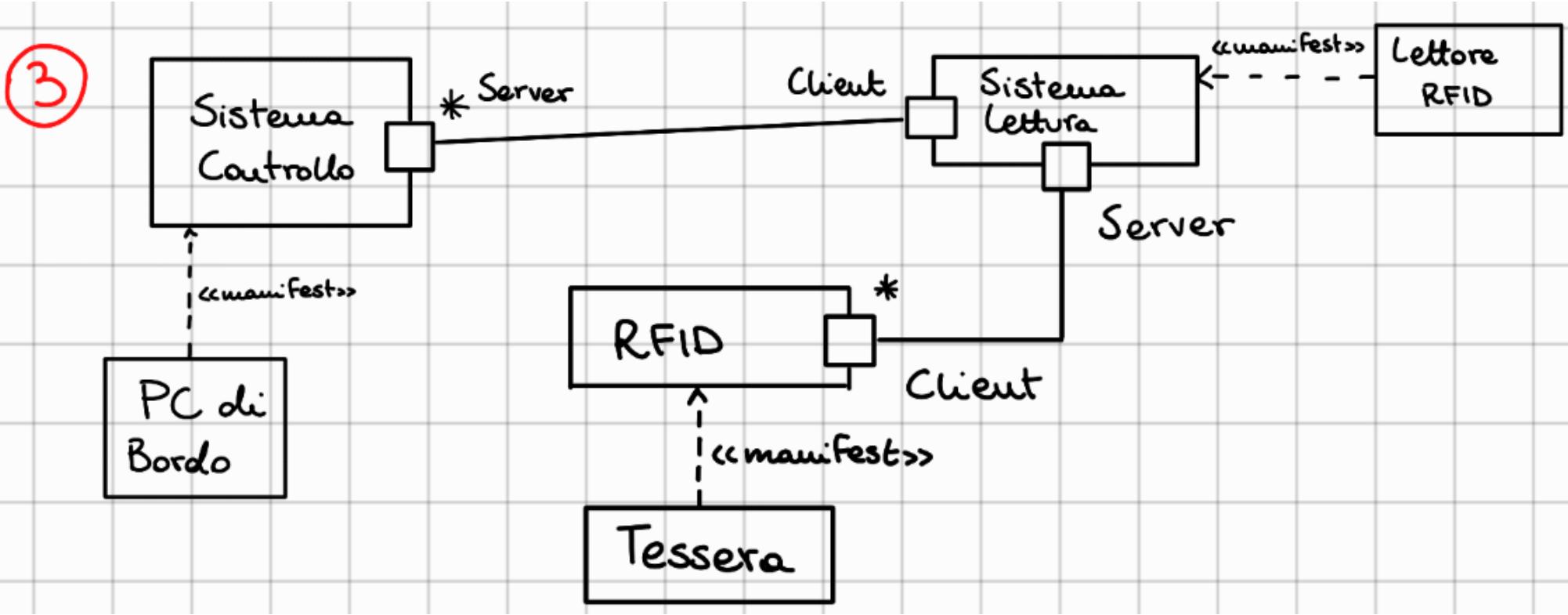
Dare un diagramma di sequenza che mostri come le componenti individuate nell'esercizio precedente implementino correttamente il seguente requisito:
Quando la hostess/lo steward manifestano l'intenzione di ripartire dopo una tappa, il sistema dà l'ok oppure indica i passeggeri mancanti, col loro numero di telefono.



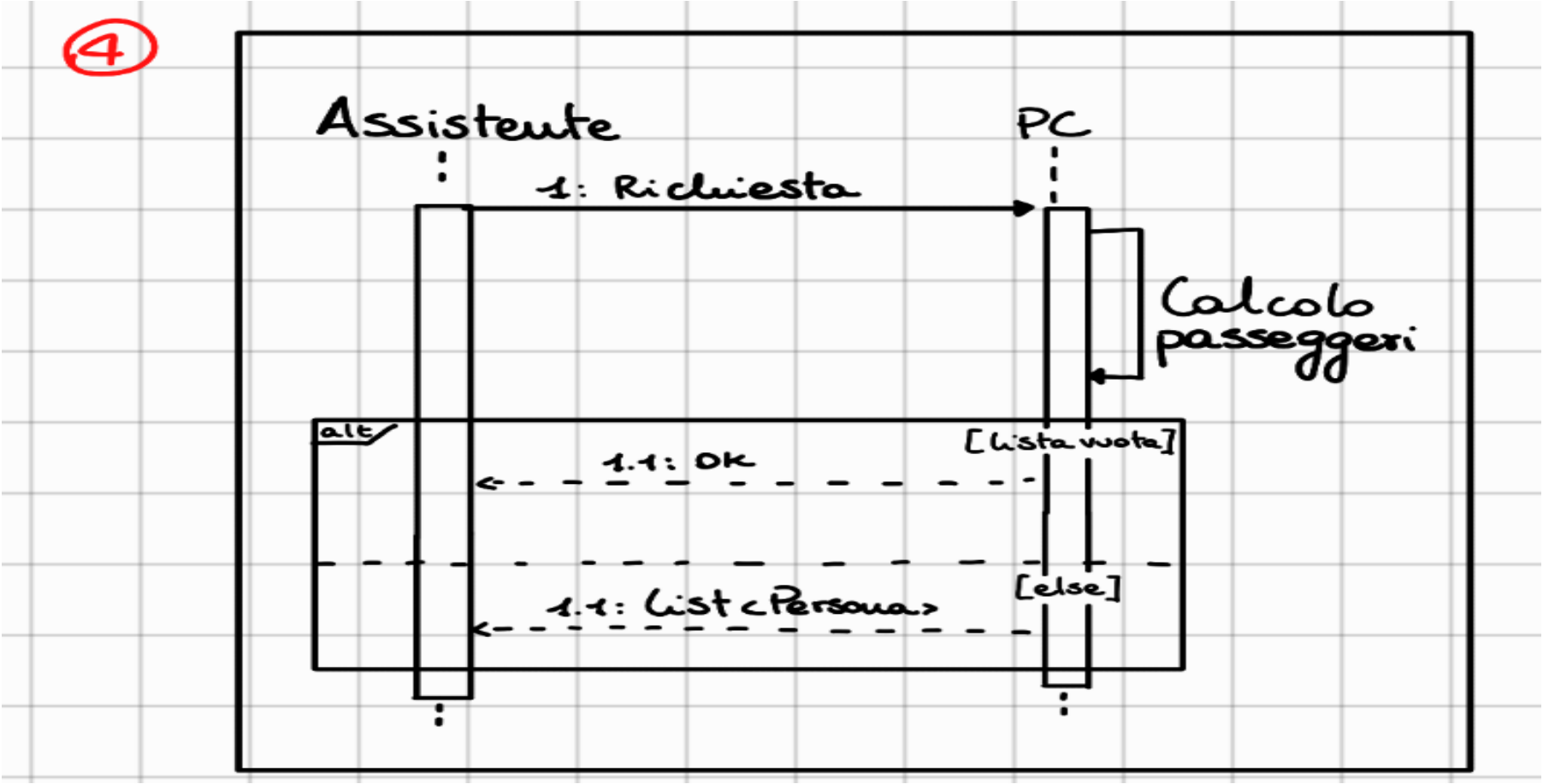
- Una lista non è una componente!!! la componente è il DB dei partecipanti
- Manca la tessera come pezzo di hw
- Manca interfaccia assistente per annunciare ripartenza/controllare assenti (?)



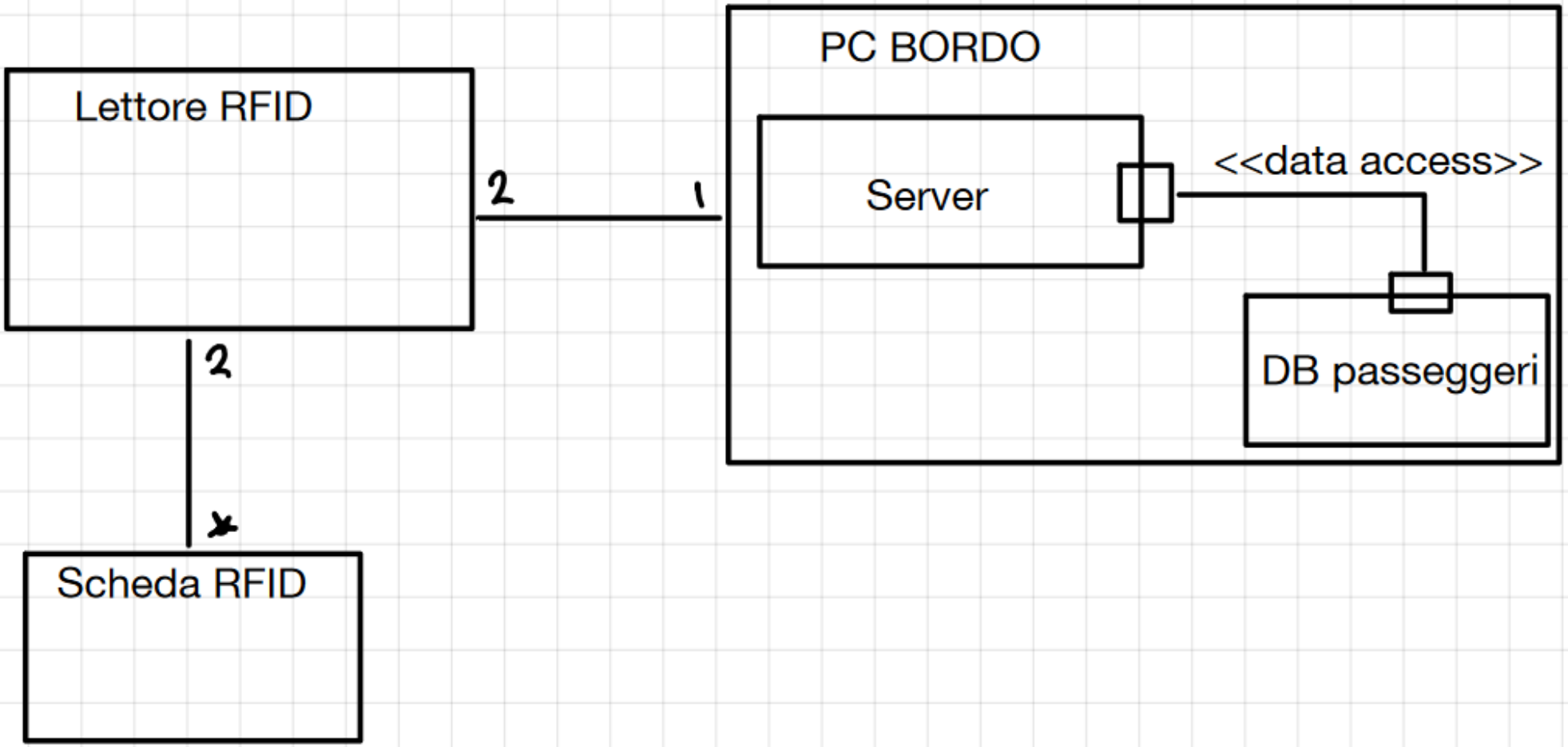
- ListaPresenza non ha senso sia paramentro



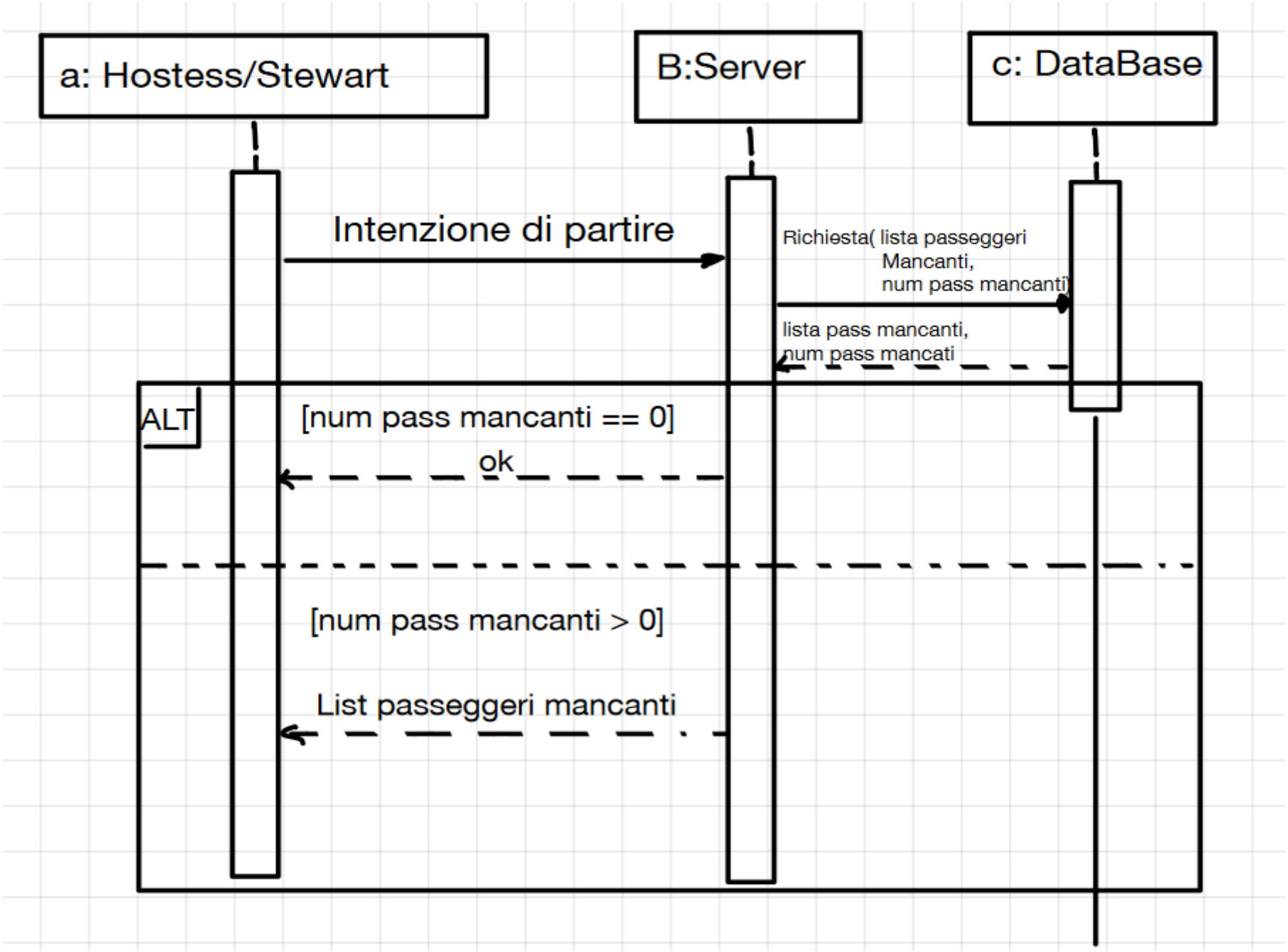
- Dislocazione sbagliata !!!
- Mancano DB e interfaccia assistente

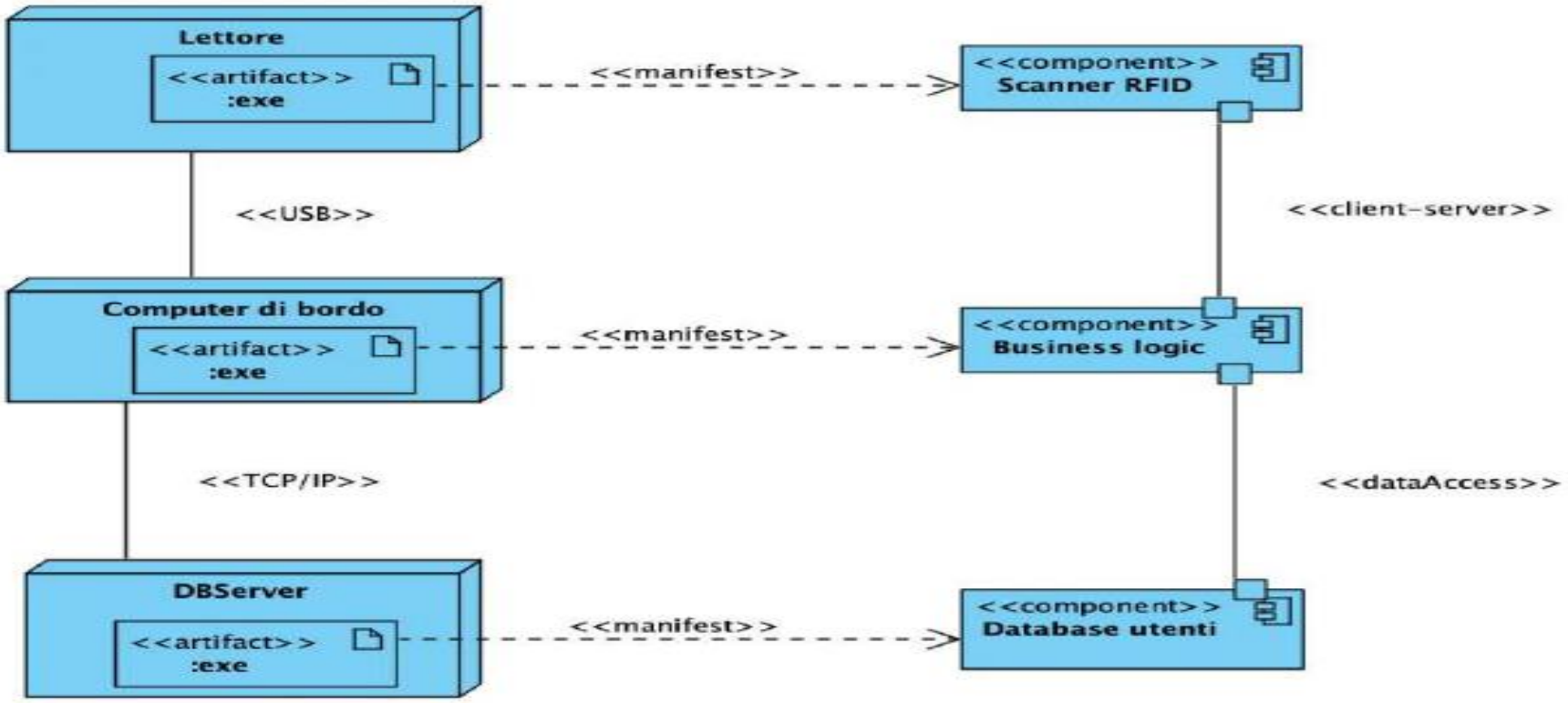


- PC non è una componente!!!!!
- Incoerente con Ex3
- Lista non è inizializzata

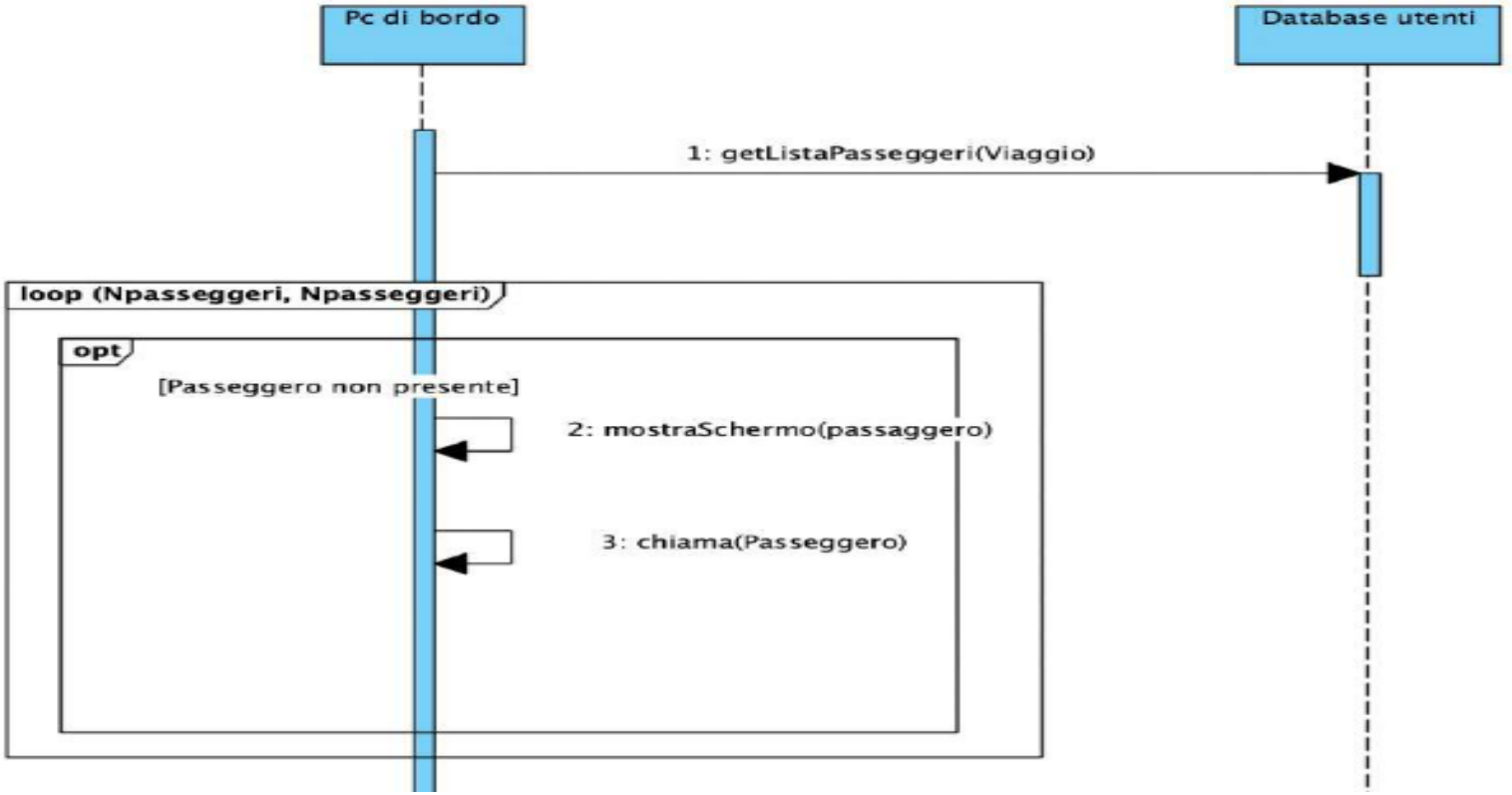


- Dislocazione sbagliata (componenti vs artefatti) !!!
- Manca interfaccia assistente
- Server è un nome molto infelice per una componente


a: Hostess/Stewart
B:Server
c: DataBase
Intenzione di partire
Richiesta(lista passeggeri
Mancanti,
num pass mancanti)
lista pass mancanti,
num pass mancati
ALT
[num pass mancanti == 0]
ok
[num pass mancanti > 0]
List passeggeri mancanti

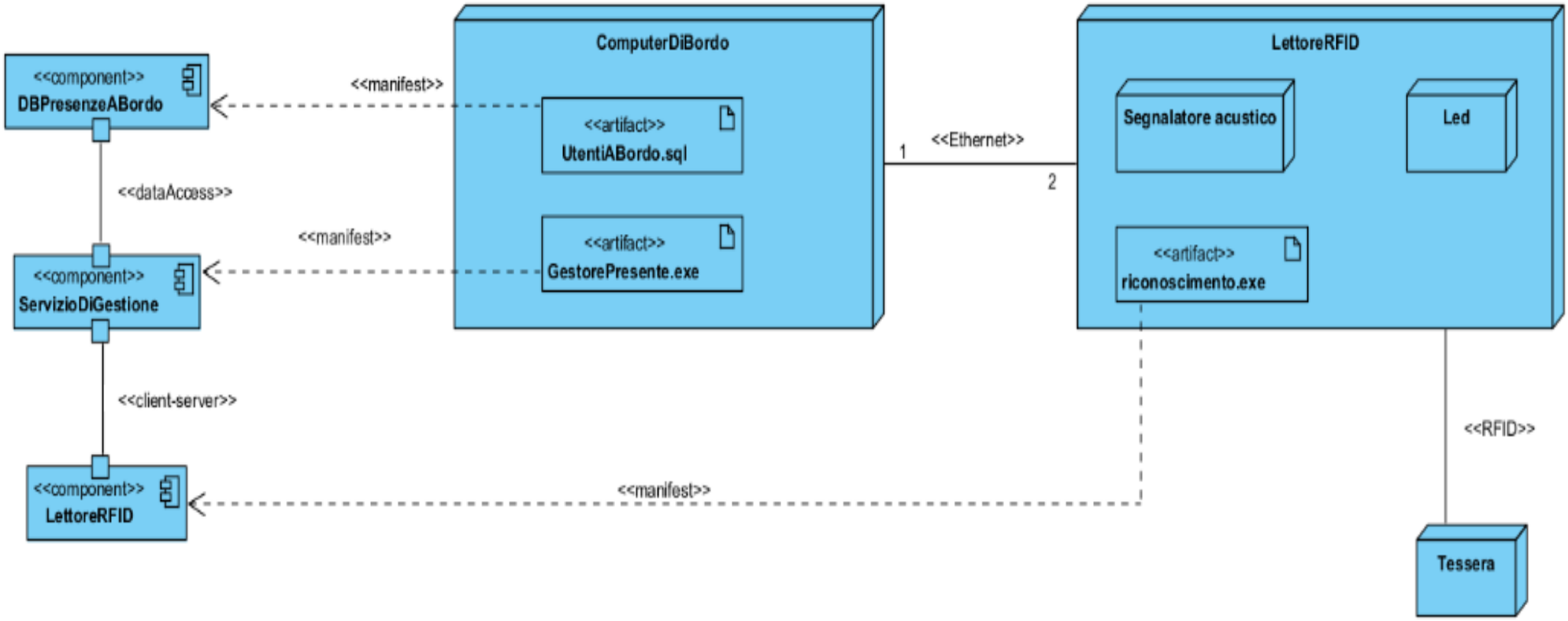


- DBServer ragionevolmente coincide col PC di bordo
- Manca interfaccia assistente



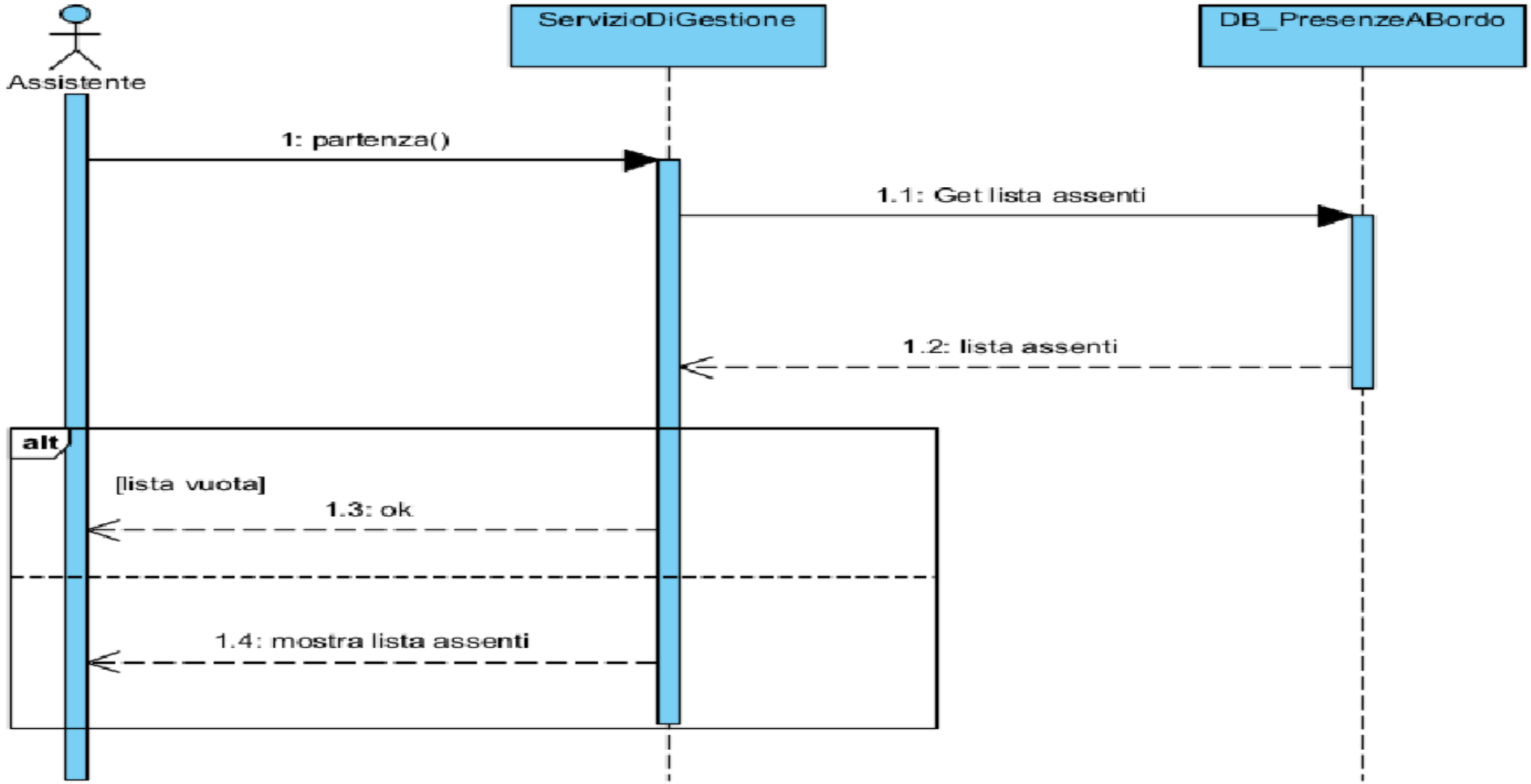
- Il sistema non chiama (ma forse coerente con casi d'uso)

3)



- Manca interfaccia assistente

4)


Assistente
ServizioDiGestione
DB_PresenzeABordo
1: partenza()
1.1: Get lista assenti
1.2: lista assenti
alt
[lista vuota]
1.3: ok
1.4: mostra lista assenti

Riassunto

- Un’interfaccia utente non si nega a nessuno
- Sull’hw si dislocano artefatti, che manifestano componenti
- Nessuno ha messo le interfacce delle componenti (a scanso equivoci, concettualmente ben diverse dalle interfacce utenti), ok non erano richieste, ma possono aiutare
- Inizializzate le variabili nei diagrammi sequenza, o almeno che si capisca
- Se si chiede un diagramma di sequenza che mostra le interazioni tra le componenti, ok un attore ma non ok hw
- I file sono artefatti, non componenti

Esercizio 5

a) Dare delle proof obligations basate sul criterio black box delle classi di equivalenza.

b) Dare il grafo di flusso del metodo controlloPartenza, sotto forma di diagramma di attività.

c) Dare un insieme di casi di test che ottenga la copertura del 100% secondo il criterio delle decisioni.

d) Definire un mutante inutile.

Date le variabili d'istanza

```
private int numPartecipanti ;

private int numPresenti = 0;

private boolean[] presente ;

// presente.length = numPartecipanti e
inizialmente presente[i] = F per ogni i
```

e la funzione

```
private void scambia(int i) {

        presente[i] = !presente[i] ;

    }
```

è stato scritto un metodo per aggiornare lo stato dei passeggeri a bordo in base al passaggio del passeggero **i**, e restituire true sse il pullman è pieno dopo il passaggio di **i**.

(Il valore del parametro **i** è generato decodificando il segnale letto dal lettore di RFID).

```
public boolean aggiornaPresenza(int i) {

  scambia(i) ;

  boolean i_entrato = presente[i] ;

  if (numPresenti == numPartecipanti-1 &
i_entrato) {

        numPresenti++;

        return true ; // tutti entrati

    }

    else {

        if (i_entrato) {

            numPresenti++ ;

            } else {

            numPresenti++ ;

            }

     return false ;

     }

    }
```

a) Proof Obligation

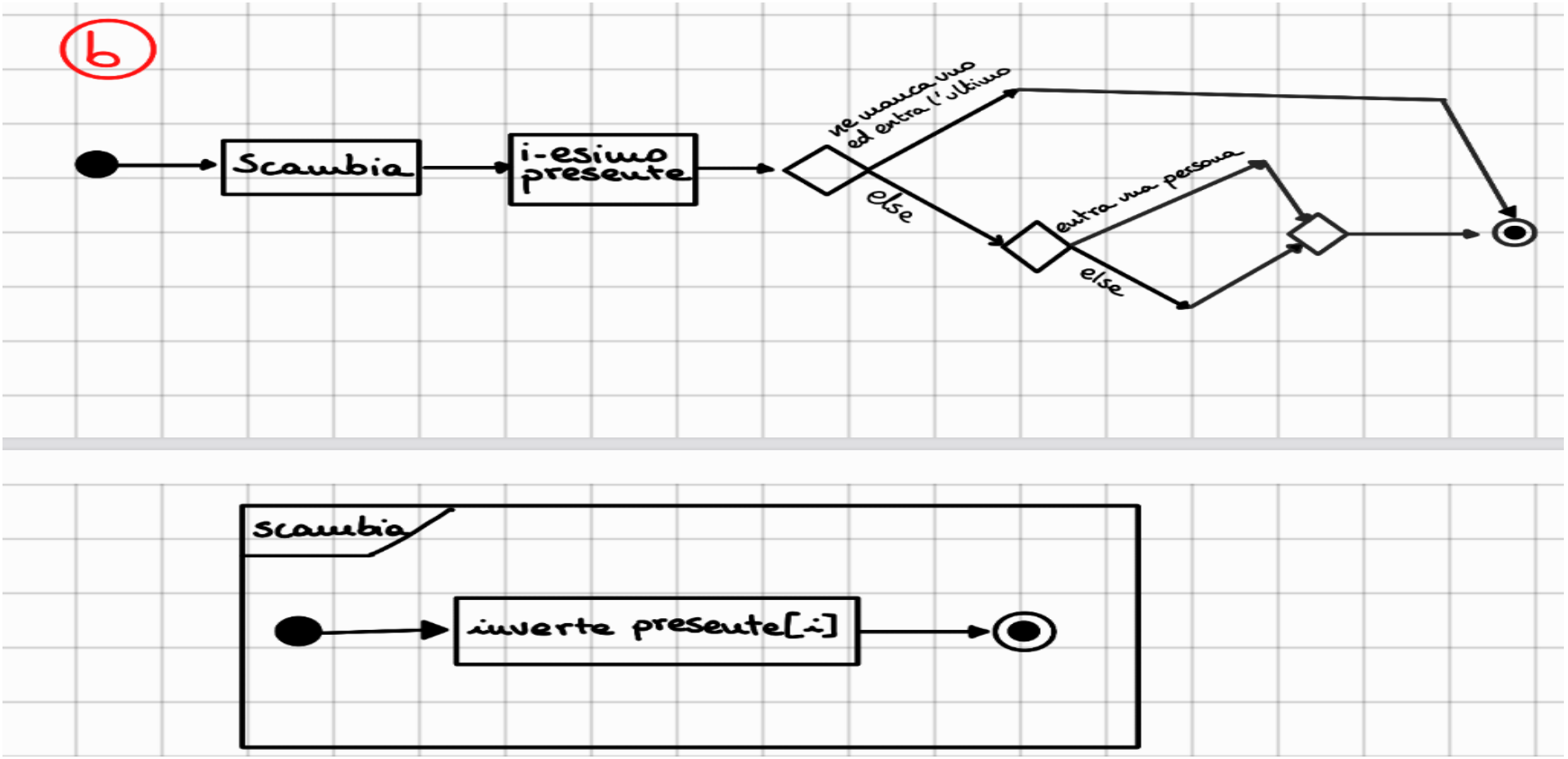
- Dando valori negativi, ci aspettiamo un errore (eccezione) con qualsiasi ambiente.
- Dando valori fuori range dell'array, ci aspettiamo un errore (eccezione) con qualsiasi ambiente
- Dando un valore in range, se il passeggero è assente, ci aspettiamo che venga aggiunto
- Dando un valore in range, se il passeggero è presente, ci aspettiamo che non venga conteggiato.

d) < 0, false, <numpartecipanti = n, numpresenti = 0, presente[n] = false >>

- Proof Obligation:
  - {un caso di test in base al numero di input inseriti}
  - {un caso di test con un numero maggiore al numero di partecipanti}
  - {un caso di test con un numero minore al numero di partecipanti}
  - {un caso di test con stesso valore ripetuto più volte}

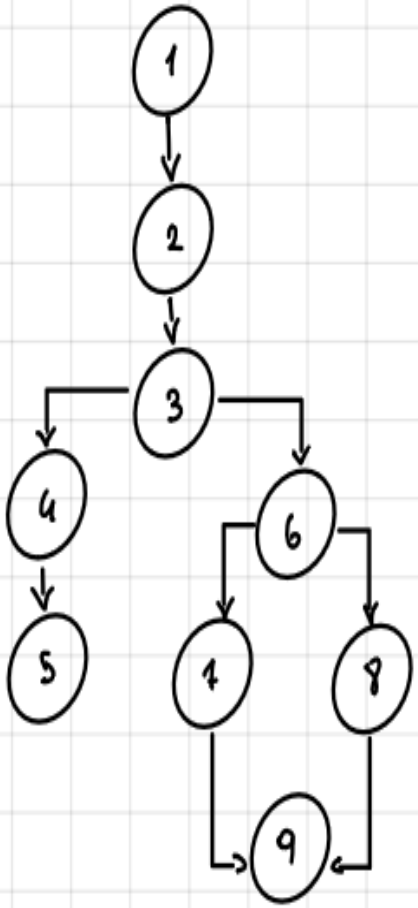
Per nulla chiaro e quindi non ok

B) Dare il grafo di flusso del metodo controlloPartenza, sotto forma di diagramma di attività.



- i_esimo presente non è un’azione
- Mancano molte azioni
- Scambia non serve qui modellarlo come sott-attività, ma ok

b)




```
public boolean aggiornaPresenza(int i) {
      1  scambia(i) ;
      2  boolean i_entrato = presente[i] ;
      3  if (numPresenti == numPartecipanti-1 & i_entrato) {
            4  numPresenti++;
            5  return true ; // è entrato l'ultimo girellone
         } else {
            6  if (i_entrato) {
                  7  numPresenti++ ;
               } else {
                  8  numPresenti++ ;
               }
            9  return false ;
         }
   }
```

- Non è un diagramma di attività !!

c) Dare un insieme di casi di test che ottenga la copertura del 100% secondo il criterio delle decisioni.

(c) 1.) {numPresenti = 5, NumPartecipanti = 6, i_entrato = true}

2.) {numPresenti = 5, NumPartecipanti = 6, i_entrato = false}

3.) {numPresenti = 5, NumPartecipanti = 8, i_entrato = true}

Il caso (1) verifica il primo if;

Il caso (2) verifica il ramo else innestato dentro il primo ramo else;

Il caso (1) verifica il ramo if innestato dentro il ramo else.

< input, output atteso, ambiente>

< 0, false, <numpartecipanti = n, numpresenti = 0, presente = false >>

< n, true, <numpartecipanti = n, numpresenti = n-1, presente = [t,t,…,f] >>

< n+1, false, <numpartecipanti = n, numpresenti = n, presente = [t,t,…,t] >>

L’ultimo dà errore

Test:

<5; true; presente[5] = false, numPresenti = 5, numPartecipanti = 6>

<5; false; presente[5] = false, numPresenti = 5, numPartecipanti = 7>

<5; false; presente[5] = true, numPresenti = 5, numPartecipanti = 7>

d) Definire un mutante inutile.

Ricordo:

inutile = ucciso da quasi tutti i casi di test

equivalente = stesso output dell’originale

d) Dentro scambia invece di

```
presente[i] = ! presente[i];
```

togliamo il not (!)

```
presente[i] = presente[i];
```

d)

```
public boolean aggiornaPresenza(int i) {
        scambia(i) ;
        boolean i_entrato = presente[i] ;
        if (numPresenti == numPartecipanti-1 & i_entrato) {
             numPresenti++;
            return true ; // è entrato l'ultimo girellone
        } else {
            if (i_entrato) {
                numPresenti++ ;
            } else {
                numPresenti++ ;
            }
            return false ;
        }
    }
```

→ numPresenti++;

Equivalente!!

Mutante:

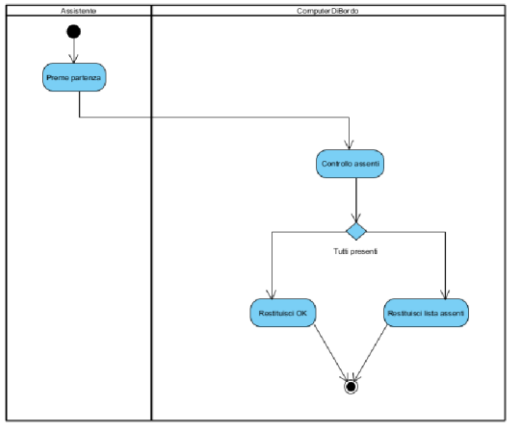
- if (!i_entrato) { numPresenti++ }

Il codice era sbagliato, per cui questo mutante è equivalente, col codice corretto sarebbe stato inutile

a) Un valore fuori dai limiti, tutti i passeggeri in entrata, tutti i passeggeri in uscita

Va bene quello che dà errore, non gli altri due casi. In ogni caso l'input è un intero, quindi la proof obligation deve considerarlo.

b)



E' un diagramma di attività quasi (mancano le guardie) corretto, ma non del metodo oggetto dell'esercizio. Le azioni del diagramma devono essere comando (o blocchi di comandi) del metodo, le decisioni le istruzioni condizionali

c) Un passeggero che entra, un passeggero che esce, un passeggero che entra riempendo il pullman.

Questa è una proof obligation, non un insieme di casi di test

d) if(numPresenti != numPartecipanti-1 …) , restituisce true ogni volta che entra un partecipante anche se non è l'ultimo

Non è inutile

Esercizio 6

a) Individuare un'ambiguità nel testo del problema, sottolineandola
b) In quali progetti conviene applicare modelli di ciclo di vita incrementali?
c) Da un punto di vista teorico, qual è il limite della verifica del sw?

6) (a) Qual è la differenza tra programma e programma di viaggio?

(b) Quando i requisiti sono stabili.

(c) Non esiste un algoritmo che dimostri la correttezza del programma

a)

In ogni sede c'è un apparecchio di scrittura di etichette RFID. Al momento dell'acquisto di un biglietto, viene prodotta una tessera, con l'identificativo del viaggio e del passeggero. Se il pagamento è stato fatto via web, la tessera è consegnata al momento del check-in.

- Chi consegna la tessera?

b)

Il modello a ciclo di vita incrementale si applica bene ai progetti dove è richiesta una funzionalità base semplice, in tempi brevi, a cui poi verranno aggiunte eventualmente altre feature.

c)

Basandoci sull'Halting Problem il limite della verifica SW teorico sta nell'impossibilità di provare se un programma termina, inoltre è impossibile provare tutti i casi di input, essendo infiniti nella maggior parte dei casi

6)

a) Non è specificato se il check-in è effettuato sul pullman dall'assistente o in sede.

b) Nei progetti in cui i requisiti sono chiari e stabili ed è necessario uscire velocemente con qualcosa o ritardare l'implementazione di funzionalità che dipendono da fattori esterni.

c) Per avere la correttezza formale di un SW abbiamo bisogno di effettuare un testing esaustivo che richiede tempo infinito

1.Ok
2.Ok
3.Ok

…in bocca al lupo per il compito

(La capitale dello Utah è Salt Lake City)