Sistema Gestione ZTL

Riccardo Maria Pesce

Anno Accademico 2019-2020

Abstract

In questa terza iterazione verrà eseguito un refactoring, in modo da adempire ad alcuni pattern GRASP ed avere un software dove le componenti sono ulteriormente modulari. In seguito, verrà complementata l'implementazione dei casi d'uso *UC3* ed *UC4*

Contenuti

1 Resoconto Iterazione 2

Sebbene il sistema funzioni, si nota che esso è difficile da mantenere, dato che:

- Il sistema centrale presenta un antipattern, visto che si presenta pieno di responsabilità
- Vengono passati parametri, anche inutili, ai costruttori (esempio i costruttori *Terminale* ed *Autenticatore*)

Il programma sembra essere molto disordinato e disorganizzato, e questo non è una cosa positiva nelle prossime fasi del progetto.
Andiamo pertanto a definire il piano di questa iterazione.

2 Piano per la Iterazione 3

In questa terza iterazione implementeremo:

- Il caso d'uso *UC3*, includendo la funzione di rimozione del terminale. La funzione di modifica, dopo aver discusso, non verrà implementata in quanto essa consta di due operazione (rimozione ed aggiunta) primitive gia implementate
- Il caso d'uso *UC4*, includendo la rimozione del residente. Per suddetti motivi, la modifica non verrà implementata
- Verranno implementati dei Controllori, per accessi, in modo che vengano adempiti i pattern *GRASP Creator, Low Coupling, High Cohesion, Expert,Controller*. Vedremo in seguito più dettagli. Precisiamo che tale scelta influenzerà solamente la fase di progettazione e non analisi.
- Verrà eseguito un refactoring dei metodi/ costruttori.

3 Requisiti

- L'impiegato deve poter registrare un nuovo terminale, munito di codice univoco, al sistema centrale il quale a sua volta assegnerà al nuovo dispositivo un profilo a seconda dei valichi e intervalli consentiti.

I profili che possono essere assegnati ad un terminale sono due:

  - Il profilo abilitato a gestire i residenti, i quali godono di accessi da tutti i valichi ed intervalli illimitati.
  - Il profilo abilitato a gestire gli utenti carico-scarico, ossia gli utenti abilitati all'accesso da un solo varco in uno o due intervalli di tempo, della durata di massimo un'ora.

  Ogni terminale deve riconoscere tali tipologie di utente per comportarsi adeguatamente.

- L'impiegato deve poter gestire (aggiungere, rimuovere) la lista di utenti residenti in modo tale che essi possano essere gestiti correttamente dai terminali. Tale lista sarà salvata in memoria persistente a fine giornata e caricata dal sistema centrale all'inizio del nuovo giorno.

4 Modello dei Casi D'Uso

Riportiamo i casi d'uso *UC3* ed *UC4* nella loro completezza.

4.1 Caso d'uso UC3 - Gestisci Terminale

Attore primario: Impiegato

- *Scenari principali di successo*
  - **Aggiungi Terminale**
    1. L'impiegato immette il codice identificativo del terminale da aggiungere.
    2. Il sistema, assicuratosi che tale codice non appartenga ad altro terminale installato, richiede all'impiegato il profilo da associare al nuovo terminale.
    3. Si possono verificare le seguenti opzioni:
       * Il profilo richiesto è *carico-scarico*. In questo caso, se non esiste un terminale con tale profilo, la richiesta viene accettata. Altrimenti, verrà chiesto di modificare il profilo dell'attuale terminale *carico-scarico* prima di procedere. Questo perchè deve essere solo uno il

terminale autorizzato al riconoscimento di tale categoria di utenti. Di norma, questo scenario dovrebbe avvenire all'inizio, prima dell'avvio del sistema, in quanto deve essere garantito almeno un terminale con questo profilo

    * Il profilo richiesto è *residente*. In tal caso, dato che non esistono limitazioni circa tale categoria, il profilo viene impostato correttamente e nessun'altra azione è richiesta

  - **Rimuovi Terminale**

    1. L'impiegato immette il codice identificativo del terminale da rimuovere.
    2. Il sistema, assicuratosi che tale esista, compie una delle seguenti azioni.
       * Se il terminale da rimuovere è di tipo *carico-scarico*, viene chiesto all'impiegato di selezionare un altro terminale per tale funzione.
       * Se il terminale da rimuovere ha profilo *residente*, viene rimosso semplicemnte, senza ulteriori azioni.

- *Scenari alternativi*

  - **Aggiungi Terminale**

    1. Se il numero inserito è gia presente, allora il sistema ritornerà un messaggio d'errore.
    2. Se, nel caso in cui si sta aggiungendo un terminale *carico-scarico* e quando viene chiesto di cambiare il corrente terminale designato a tale funzione, viene fornita una risposta non affermativa o sbagliata, il sistema ritorna un messaggio d'errore.

  - **Rimuovi Terminale**

    1. Se il codice identificativo non esiste, viene ritornato un messaggio d'errore.
    2. Se tale è l'unico terminale attivo nel sistema, viene respinta la richiesta con un apposito messaggio d'errore.

3. Se, nel caso in cui si vuole rimuovere il terminale *carico-scarico*, ed alla richiesta di inserire il codice di un altro terminale per trasferire tale profilo viene inserito un codice inesistente, viene restituito un messaggio d'errore e terminata l'esecuzione.

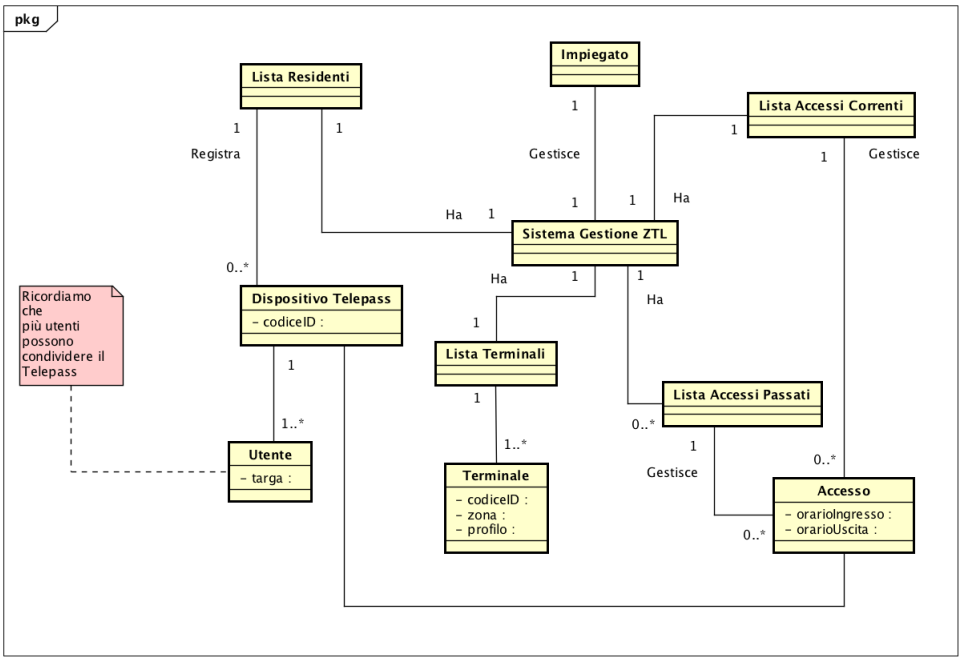
4.2 Caso d'uso UC4 - Gestisci Residente

Attore primario: Impiegato

- *Scenari principali di successo*
  - **Aggiungi Residente**
    1. L'impiegato immette i dati anagrafici, ed assegna al residente il codice identificativo del dispositivo telepass posseduto dall'utente stesso.
    2. Il sistema, assicuratosi che l'utente non esiste gia (attraverso il controllo del codice identificativo), lo registrerà con successo.
  - **Rimuovi Residente**
    1. L'impiegato immette il codice identificativo dell'utente da rimuovere.
    2. Il sistema, assicuratosi che l'utente esista, lo rimuove senza ulteriori azioni.
- *Scenari alternativi*
  1. **Aggiungi Residente**
     (a) Se l'utente gia esiste, la richiesta verrà semplicemnte ignorata con un messaggio d'errore.
  2. **Rimuovi Residente**
     (a) Se l'utente non esiste, la richiesta verrà semplicemnte ignorata con un messaggio d'errore.

5 Modello di Dominio

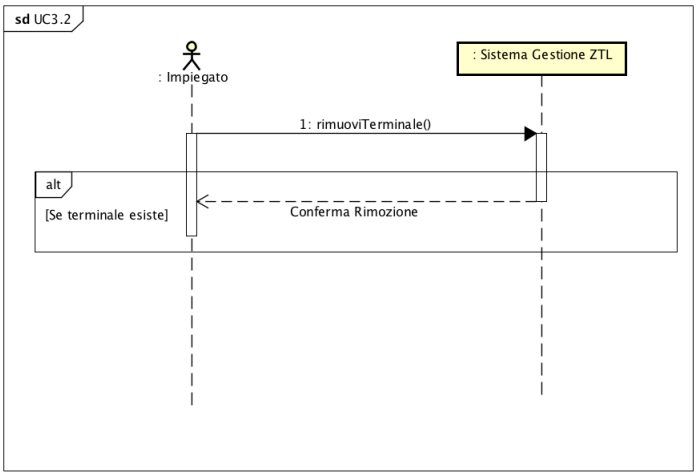
L'analisi di dominio identifica le classi concettuali identificate nella precedente iterazione.



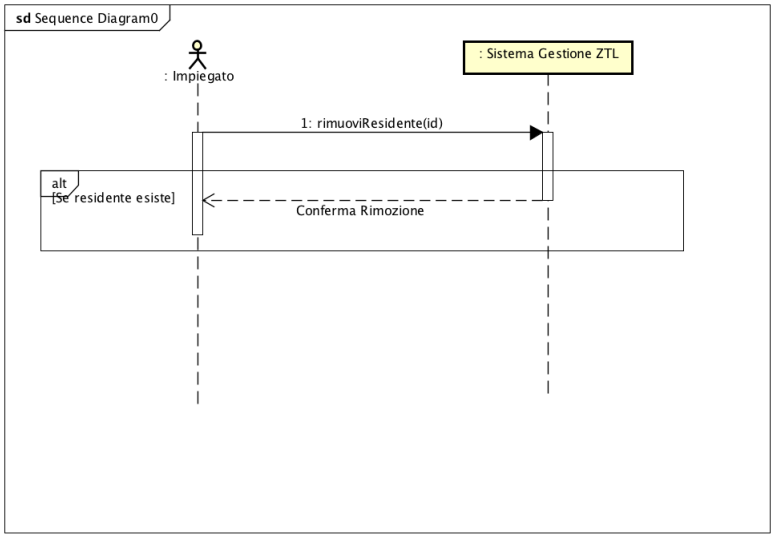
6 Diagrammi di Sequenza di Sistema

6.1 UC3.2 *(Rimuovi Terminale)*

Dato che implementeremo due varianti dell'operazione `rimuoviTerminale` (in quanto si può rimuovere sia per id che per zona), nel diagramma si omettono volutamente i parametri.



6.2 UC4.2 *(Rimuovi Residente)*

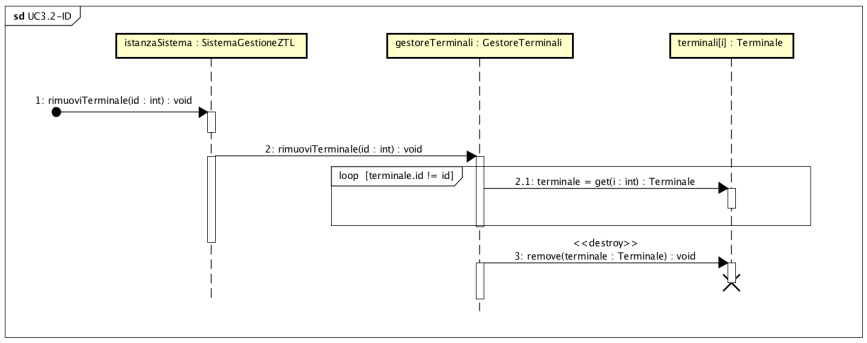


7 Diagrammi di Interazione

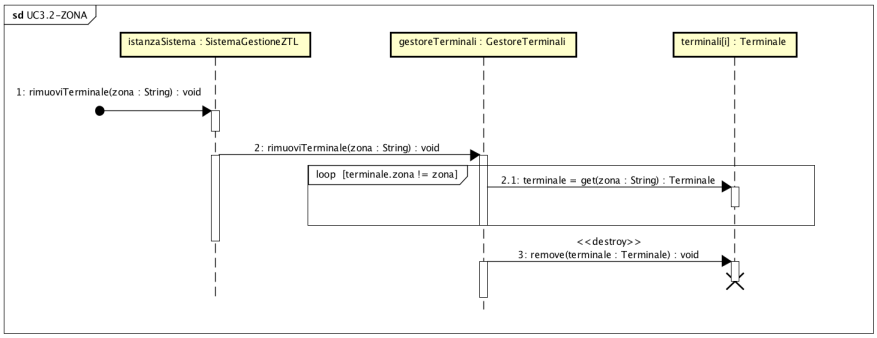
Innanzitutto riportiamo i diagrammi di interazione dei casi d'uso introdotti in questa iterazione, ed in seguito vogliamo riportare anche i diagrammi dei casi d'uso definiti in precedenza, rifiniti, in modo da portarci al refactoring in maniera più agevole.

7.1 UC3.2 - Rimuovi Terminale

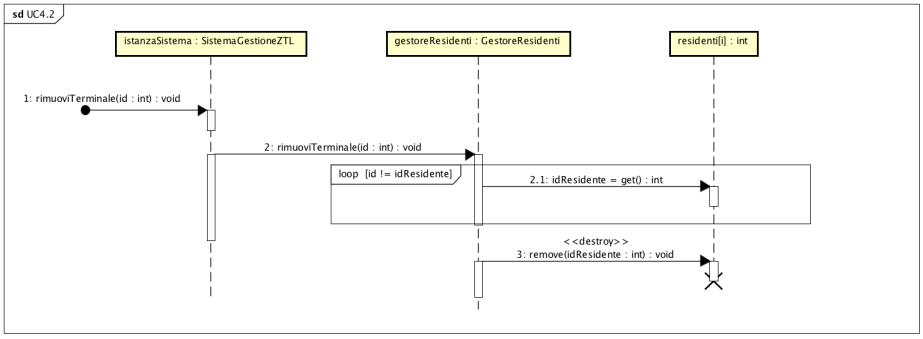
Nel caso in cui vogliamo rimuovere il terminale per *id*, il diagramma si presenta così:



Mentre se vogliamo rimuovere il terminale per zona:



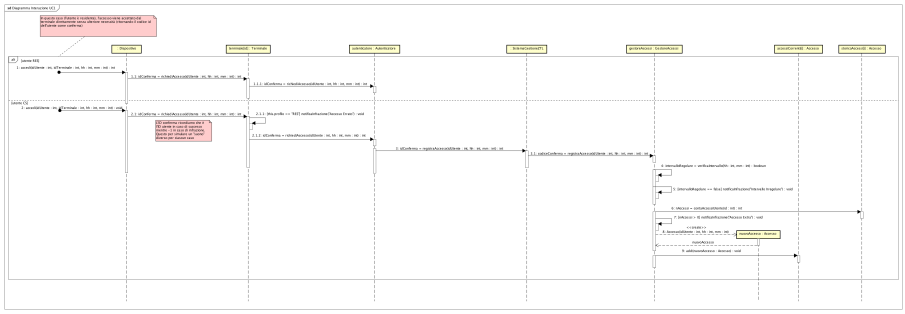
7.2 UC4.2 - Rimuovi Residente



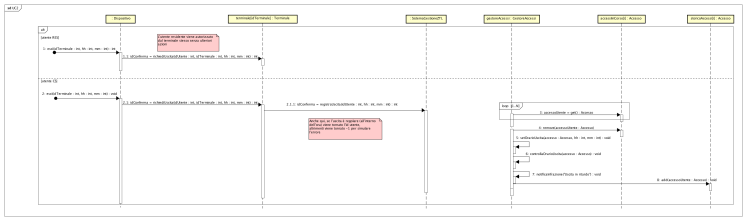
8 Diagrammi di Interazione refactored

Questi diagrammi afferiscono alla precedente iterazione, con le opportune modifiche apportate in tale iterazione.

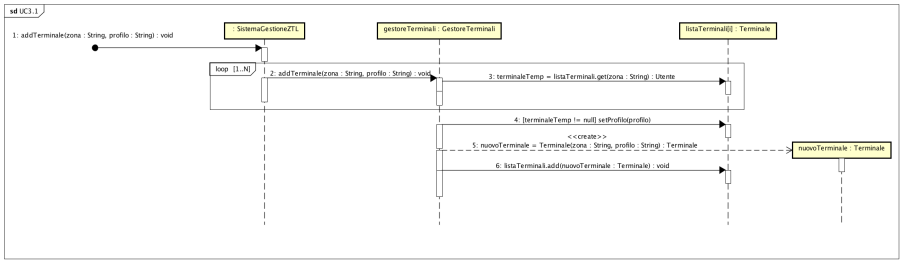
8.1 UC1



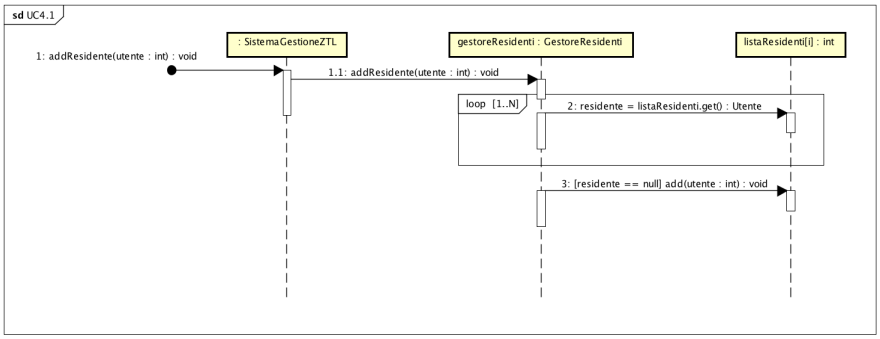
8.2 UC2



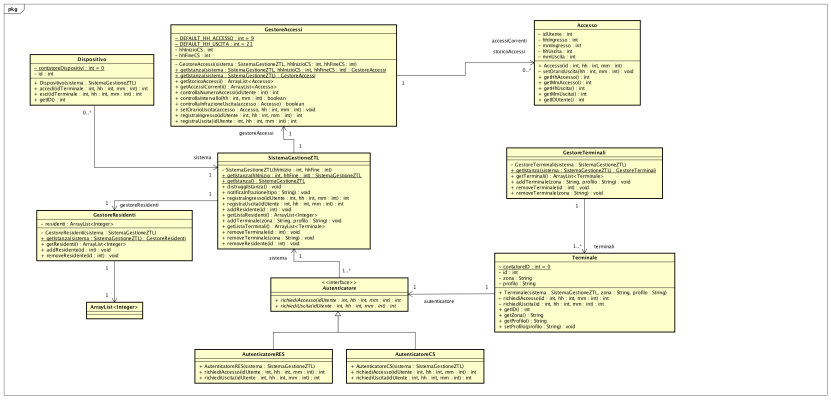
8.3 UC3.1 *Aggiungi Terminale*



8.4 UC4.1 *Aggiungi Residente*



9 Diagramma delle Classi



10 Motivazioni e Scelte Progettuali

10.1 Applicazione Pattern *GRASP*

Il sistema applica i pattern *GRASP* nel seguente modo:

- Il sistema centrale è un *Controller* in quanto riceve gli input e li delega alle parti del sistema interessate (i gestori, nel nostro caso)
- Viene ridotto l'accoppiamento *(Low Coupling)* eliminando le dipendenze tra i vari componenti e permettendo comunque una loro

comunicazione attraverso il sistema centrale (la classe *SistemaGestioneZTL*)

- La coesione viene aumentata *(High Cohesion)* grazie alle nuove classi
- In particolare, in tal caso, si applica il pattern *Pure Fabrication* dato che le varie classi *gestore* non hanno una controparte reale se non la funzionalità di modularizzare e disaccoppiare il sistema centrale dalle varie mansioni specifiche (di gestire terminali, accessi e quant'altro)
- Il pattern *Creator* viene applicato nell'istanziamento degli accessi
- L'autenticatore è accoppiato con il sistema centrale, e pertanto leggera i residenti dal *GestoreResidenti* attraverso esso. Peranto abbiamo rimosso il pattern *Observer*

10.2 Applicazione Pattern *GoF*

I pattern *GoF* sono i seguenti:

- Il pattern *Singleton*, non solo per il sistema centrale, ma anche per i vari gestori
- Il pattern *Strategy*, per la strategia di autenticazione (per utenti residenti e utenti carico-scarico)
- Come ribadito nel precedente paragrafo, il pattern *Observer* è stato rimosso in quanto ogni qualvolta un autenticatore necessita di accedere alla lista residenti, esso farà una chiamata all'API che il *Sistema Gestione ZTL* fornisce. Esso infatti non solo è un controller per l'utente, ma anche per i diversi moduli del sistema
- La classe *SistemaGestioneZTL* è una potenziale candidata per il pattern *Mediator*, in quanto gia sta assolvendo alla funzione di mediare fra le diverse componenti del sistema, disaccoppiandole. Tuttavia per ora non abbiamo fornito un meccanismo per assolvere a tale funzione in maniera standard (aggiungendo/rimuovendo) dinamicamente componenti a cui mediare.

11 Casi di Test

11.1 Test Unitari

I test unitari vanno applicati ai metodi `controllaNumeroAccessi()`, `controllaIntervallo()`, `controllaInfrazioneUscita()` della classe *GestoreAccessi*. Una volta inizializzato il sistema, procediamo ai singoli test.
Per la funzione `controllaNumeroAccessi()` abbiamo:

1. Viene fatto accedere un utente una volta. Tale utente avrà ID 5 e sarà carico-scarico
2. Viene fatto uscire
3. Viene chiamato il metodo con il suo ID e viene verificato che il valore tornato sia 1 (il numero di accessi eseguiti)
4. Il medesimo metodo viene chiamato con un altro ID utente (1 nel nostro caso). Questo dovrà tornare 0

Per la funzione `controllaIntervallo()` invece abbiamo il seguente caso di test:

1. Viene controllato se l'orario 9:00 ricade nell'intervallo, aspettandoci che l'output sia positivo (l'accesso, ricordiamo, è permesso dalle 9 alle 21 per gli utenti carico-scarico)
2. Stessa cosa per l'orario 21:00 (con medesimo risultato)
3. Viene testato l'orario 8:59 con esito negativo
4. Viene testato l'orario 21:01 con esito negativo

Per la funzione `controllaInfrazioneUscita()` invece abbiamo il seguente caso di test:

1. Creiamo un accesso che dura dalle 11:00 alle 12:00. Questo darà `false`
2. Creiamo un accesso dalle 11:00 alle 12:01. Questo darà `true`

11.2 Test unitari per la classe `Dispositivo`

Testiamo i metodi `accedi()` ed `esci()`. In particolare, per il primo:

- Istanziamo 4 dispositivi (con ID che varia tra 0 e 3 e nomi d1, d2, d3, d4)
- Aggiungiamo due terminali, uno in zona "a" profilo "CS", e l'altro in zona "b" profilo "RES"

Adesso passiamo al test:

1. d1 accede alle 8:59, e questo ritorna -1 (l'accesso per utenti carico-scarico è dalle 9 alle 21)
2. d2 accede alle 9:00, e questo ritorna 1 (l'ID di d2) in quanto in intervallo
3. d1 accede, ma questo ritorna 0 in quando d1 è ancora dentro e non può accedere una seconda volta (in tal caso si segna come 0)
4. d3 accede alle 21, e questo ritorna 2 (l'ID di d3)
5. viene fatto uscire d1 scartando il valore d'uscita (che testeremo nel secondo metodo)
6. d1 ri-accede alle 15:00, e questo da -1 in quanto ha gia effettuato l'accesso
7. d4 accede dal terminale RES alle 12, questo ritorna -1 in quanto non è residente
8. d4 diventa residente, e viene fatto accedere due volte, una dal terminale 0 ed una dal terminale 1 (senza uscita). In entrambi i casi viene ritornato l'ID. Questo perchè gli utenti residenti non vengono "tracciati", ed i loro accessi non vengono registrati

Per il secondo metodo:

- Istanziamo solo due dispositivi
- Aggiungiamo gli stessi terminali del caso di test precedente

I passi eseguiti sono i seguenti:

1. d1 accede (il risultato viene scartato) alle 12:00
2. d1 esce alle 13, con successo (viene tornato il suo ID) in quanto all'interno dei 60 minuti
3. d1 accede alle 22 (risultato ignorato)
4. d1 esce alle 22:01, con errore (-1) in quanto fuori intervallo

11.3 Esecuzione del Test Driver

Attraverso il test driver, vengono eseguiti i test che valutano il sistema nella sua completezza a fine iterazione: non testiamo le varie unità, ma il programma nella sua completezza. In particolare, queste sono le istruzioni eseguite:

1. Inizializzato il sistema con intervallo carico-scarico (9-21)
2. Creato terminale in zona "a" con profilo CS
3. Creato terminale in zona "b" con profilo RES
4. Aggiunto dispositivo di ID 0
5. Aggiunto dispositivo di ID 1
6. Dispositivo di ID 0 diventa residente
7. Dispositivo 0 entra alle 7:00 con successo
8. Dispositivo 0 esce alle 9:00 con successo
9. Dispositivo 1 entra alle 10:00 con successo
10. Dispositivo 1 esce alle 10:30 con successo
11. Dispositivo 1 entra alle 22:00 con due fallimenti: "Ingresso Extra" e "Intervallo Irregolare"
12. Dispositivo 1 esce alle 20:00, con errore in quanto non esiste accesso attivo associato ad esso