

Ingegneria del Software A.A. 2016/2017

Esame 2017-04-18

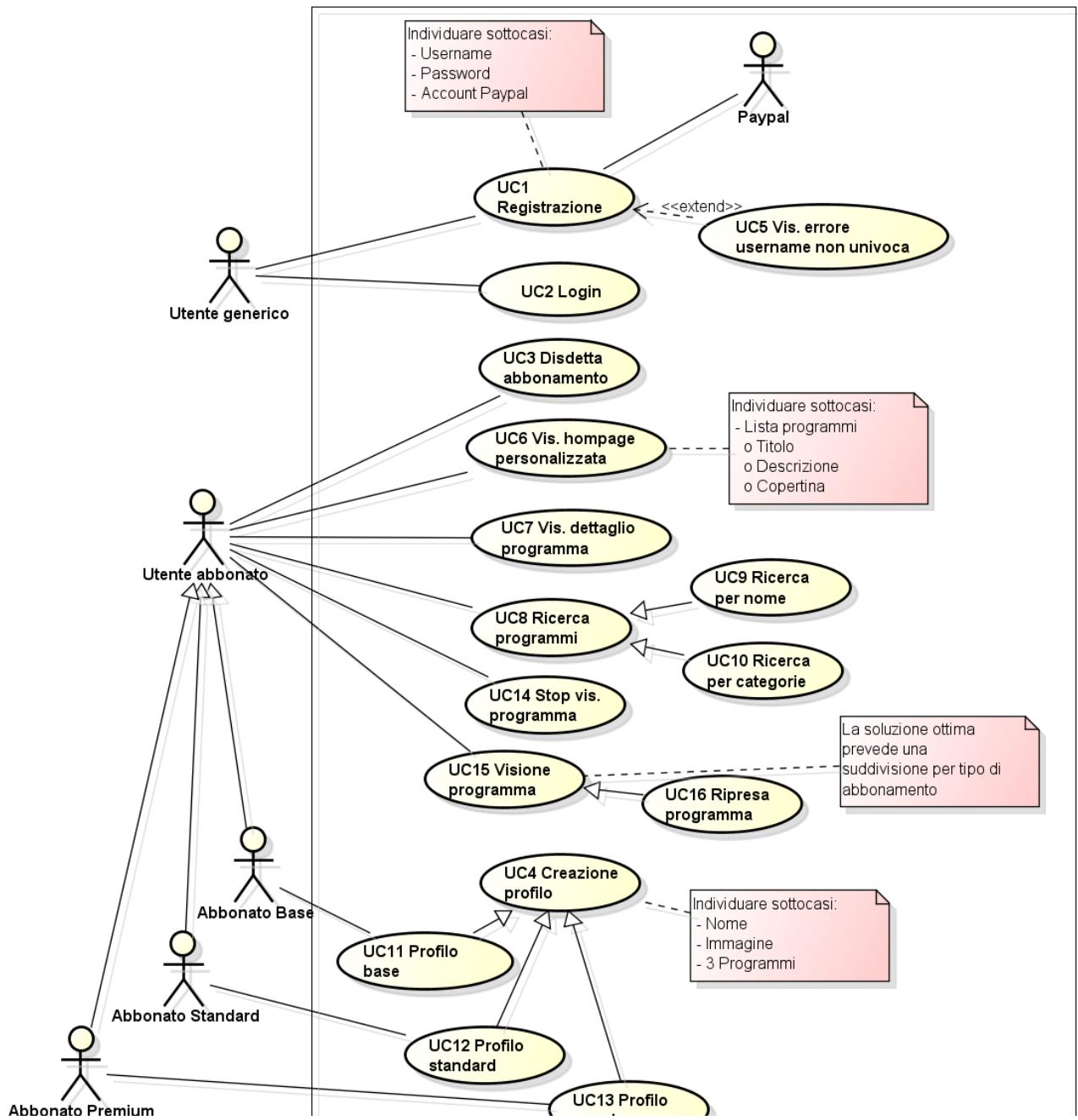
Esercizio 1 (6 punti)

Descrizione

Netflix è una piattaforma di streaming video presente oramai in gran parte del mondo. La sua particolarità è quella di supportare una miriade di piattaforme differenti per lo *streaming*: dagli smartphone alle televisioni. L'account da utilizzare per visualizzare i contenuti è unico. La registrazione richiede una username univoca, una password e l'inserimento di un conto Paypal valido. Il primo mese di visione è gratuito. Dal secondo mese in poi, un utente può disdire l'abbonamento in ogni momento, senza che gli venga addebitata alcuna penale. Esistono tre piani di abbonamento: base in *standard definition*, che consente di creare un solo profilo; standard in *high definition*, che consente di associare due profili al proprio account; *premium* in 4k, che consente di associare fino a 4 profili. Un profilo è contraddistinto da un nome, un'immagine e la scelta di 3 programmi dal catalogo Netflix, che costituiscono le preferenze base dell'utente. Con queste informazioni, il sistema crea una homepage personalizzata, che visualizza i programmi consigliati. Ogni programma in questa lista riporta il titolo, una brevissima descrizione ed un'immagine di copertina. Entrando nel dettaglio, invece, viene fornito anche un ranking dato dagli utenti (da 1 a 5 stelle), la definizione disponibile e l'eventuale scelta dell'episodio da visualizzare. Netflix consente anche di ricercare i programmi nel proprio catalogo, navigando per categorie predefinite, o effettuando una ricerca per nome. Infine, durante la visione di un programma, è possibile fermare la riproduzione. Nel momento in cui l'utente deciderà di riprendere la visione del medesimo programma, questa riprenderà dall'ultimo punto nel quale era stata precedentemente interrotta.

Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari descritti. Non è richiesta la descrizione testuale dei casi d'uso individuati.

Soluzione



Esercizio 2 (7 punti)

Descrizione

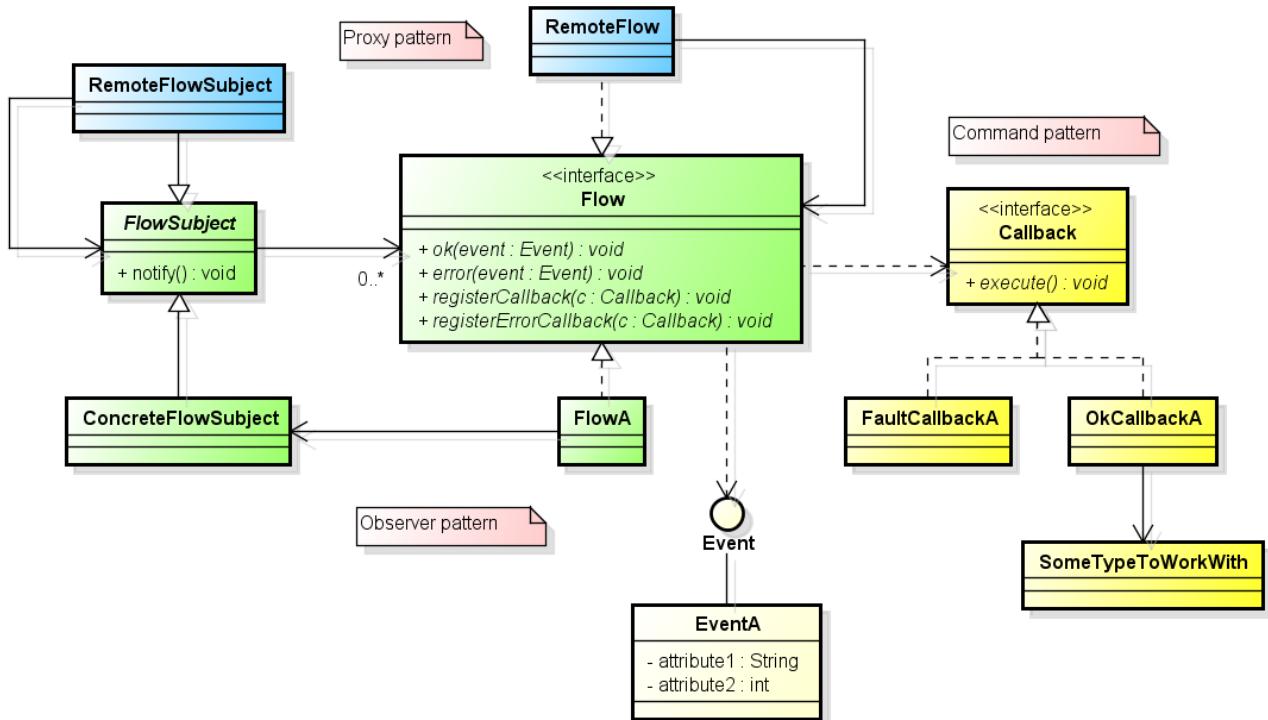
Da qualche anno si parla sempre più insistentemente di *reactive programming*. In questo stile di programmazione un oggetto non richiede ad un altro oggetto di eseguire delle operazioni, ma reagisce ad eventi. Esistono diverse librerie, che permettono di utilizzare questo stile di programmazione. Solitamente portano nel nome l'acronimo Rx. Si immagini di dover sviluppare una libreria del tipo Rx. È necessario fornire un meccanismo attraverso il quale un oggetto possa osservare eventi scatenati da un altro oggetto. Si chiami il tipo che permette di osservare un evento, *Flow*. Un client che voglia utilizzare un oggetto di tipo *Flow*, deve poter registrare essenzialmente due *callback*: una in caso di eccezione del flow ed una in

caso di evento osservato correttamente. Le *callback* possono implementare qualsiasi operazione sui dati relativi all'evento osservato. Si fornisca quindi un'opportuna struttura. Infine, deve essere possibile osservare sia eventi locali, sia eventi scatenati in remoto. Si fornisca un'adeguata struttura.

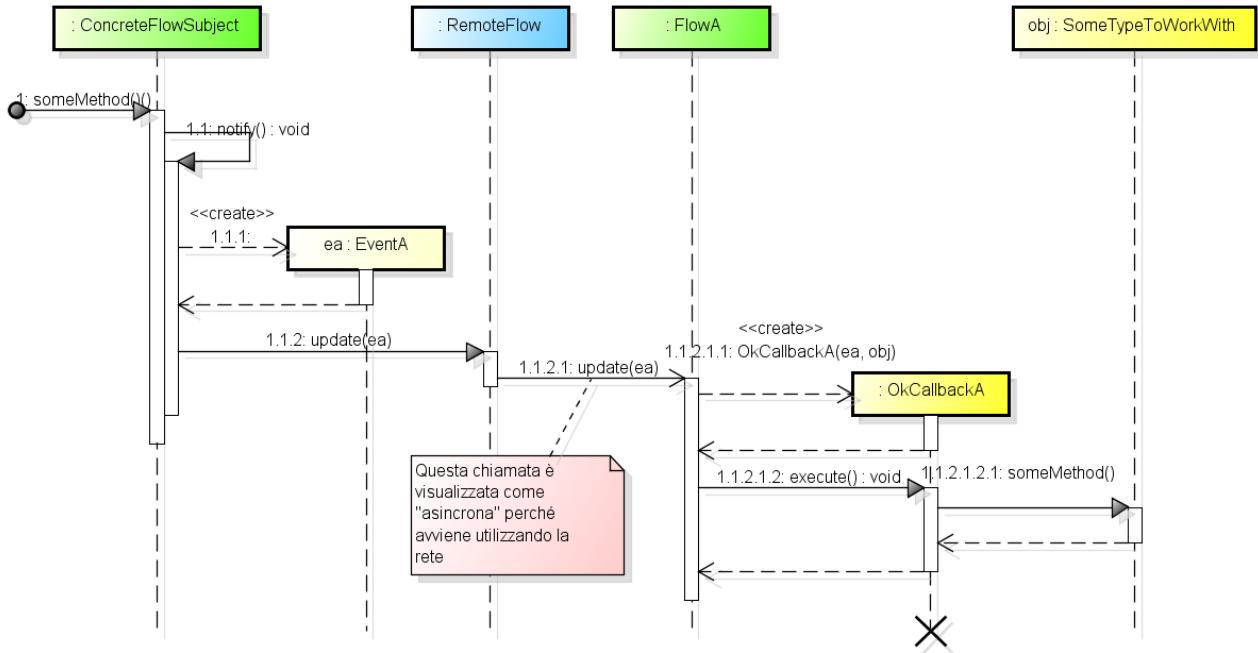
Si modelli il sistema descritto utilizzando un diagramma delle classi e gli opportuni *design pattern*. Inoltre, si descriva utilizzando un diagramma di sequenza la ricezione di un evento di tipo *EventA* da remoto, che scateni un'operazione generica attraverso una *callback* non di errore.

Soluzione

Una possibile soluzione prevede l'utilizzo dell'Observer pattern, del Command pattern e del Proxy pattern.



Segue il diagramma di sequenza richiesto.



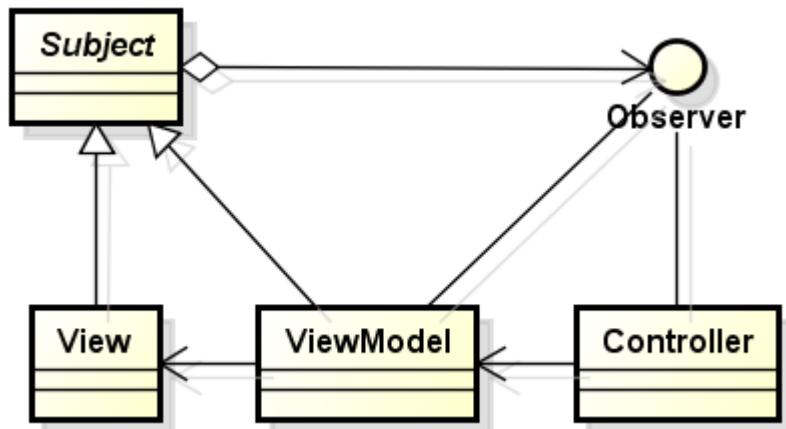
Esercizio 3 (3 punti)

Descrizione

AngularJs è un framework Javascript per lo sviluppo di applicazioni web che implementa il pattern Model-View-ViewModel. Questo pattern permette di tenere sempre sincronizzati i seguenti tre elementi di un'applicazione: una vista con un oggetto `$scope` (view-model) e quest'ultimo con un oggetto di tipo controller. Utilizzando l'opportuno design pattern, si fornisca un diagramma delle classi che modelli in modo opportuno il funzionamento semplificato descritto di AngularJs.

Soluzione

La soluzione prevede almeno una doppia applicazione del pattern Observer. La soluzione ottima è però più complessa, perché View-ViewModel e ViewModel-Controller partecipano ad un *loop* costante di aggiornamenti.



Ingegneria del Software A.A. 2016/2017

Esame 2017-05-15

Esercizio 1 (6 punti)

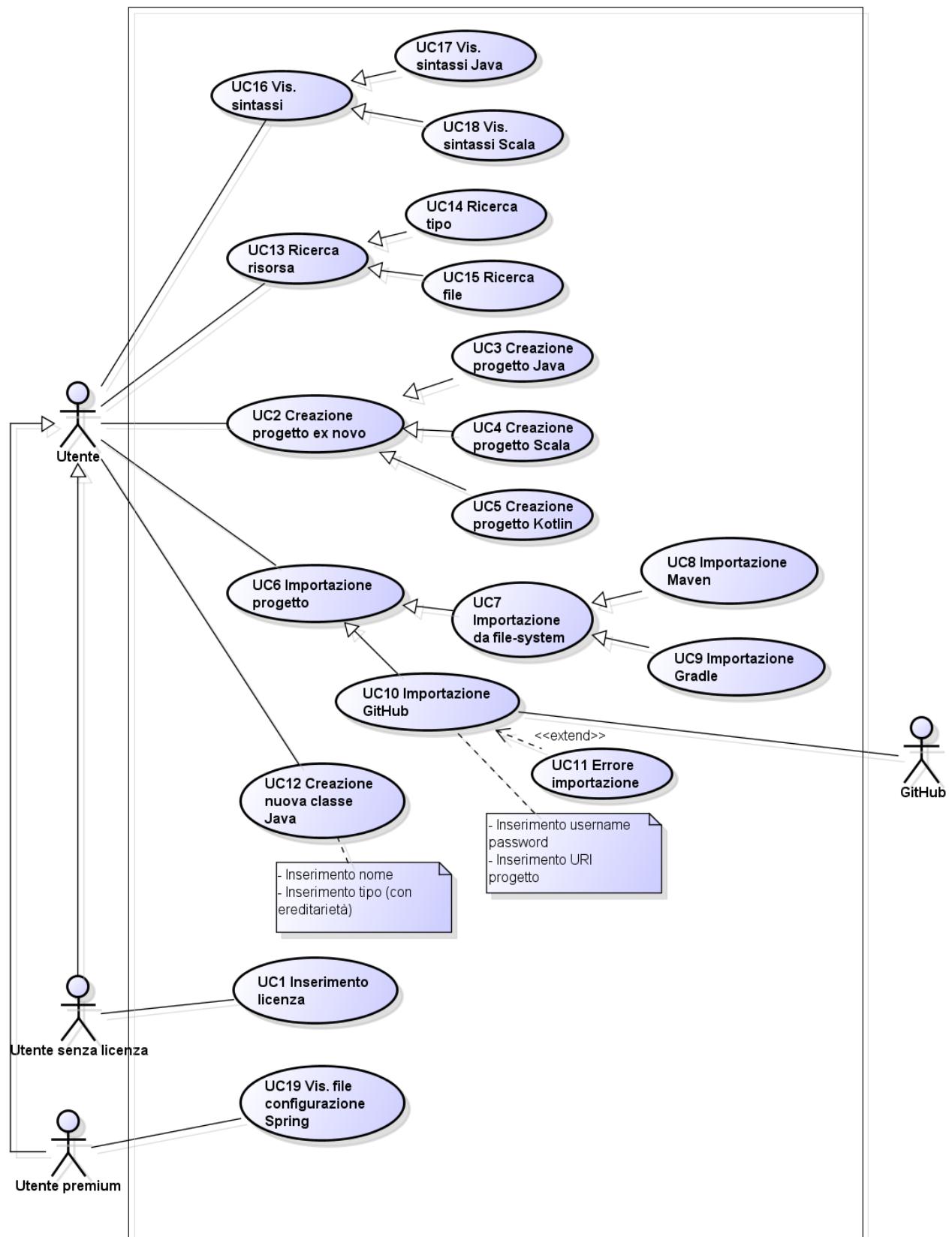
Descrizione

Uno dei più famosi programmi IDE per lo sviluppo Java è indubbiamente IntelliJ IDEA. Mentre nei primi anni di vita il software era esclusivamente a pagamento, ora, essendo l'IDE di riferimento per lo sviluppo di applicazioni Android, prevede anche una versione CE non a pagamento. Le due versioni differiscono fra loro per le funzionalità offerte. Per accedere alla funzione premium è necessario disporre di una licenza valida, da inserire opportunamente all'interno del programma. Gli utenti non a pagamento possono di base creare diversi tipi di progetto: Java, Scala, Kotlin, ecc... E' possibile scegliere se creare un progetto *ex novo*, se importarlo da un progetto Maven o Gradle esistente su filesystem o se importarlo dal repository esterno GitHub. In questo caso, è necessario effettuare l'autenticazione al proprio account GitHub (inserendo username e password) e fornire l'URI del progetto da clonare in locale. La creazione di una nuova classe Java richiede l'inserimento del Nome della classe e del suo tipo (classe, interfaccia o enumerazione). La sintassi all'interno di un file viene evidenziata opportunamente dal programma, sia essa Java, Scala, ecc... Sono disponibili diversi tipi di ricerca delle risorse all'interno dell'IDE. La pressione di Ctrl+N permette di cercare direttamente tra i tipi (classi) del linguaggio scelto, mentre la pressione di Maiusc+Maiusc permette la ricerca di qualsiasi tipo di risorsa gestita. Gli utenti Premium, hanno un'integrazione maggiore con alcuni framework Java, come ad esempio con Spring. Nella versione Premium è possibile infatti risalire al file di configurazione Spring per una classe Java direttamente selezionandola.

Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari descritti. Non è richiesta la descrizione testuale dei casi d'uso individuati.

Soluzione

La soluzione fa ampio uso dell'ereditarietà fra casi d'uso. Ove riportate le caratteristiche dei sottocasi d'uso come commento, è necessario effettivamente fornirli come diagramma.



Esercizio 2 (7 punti)

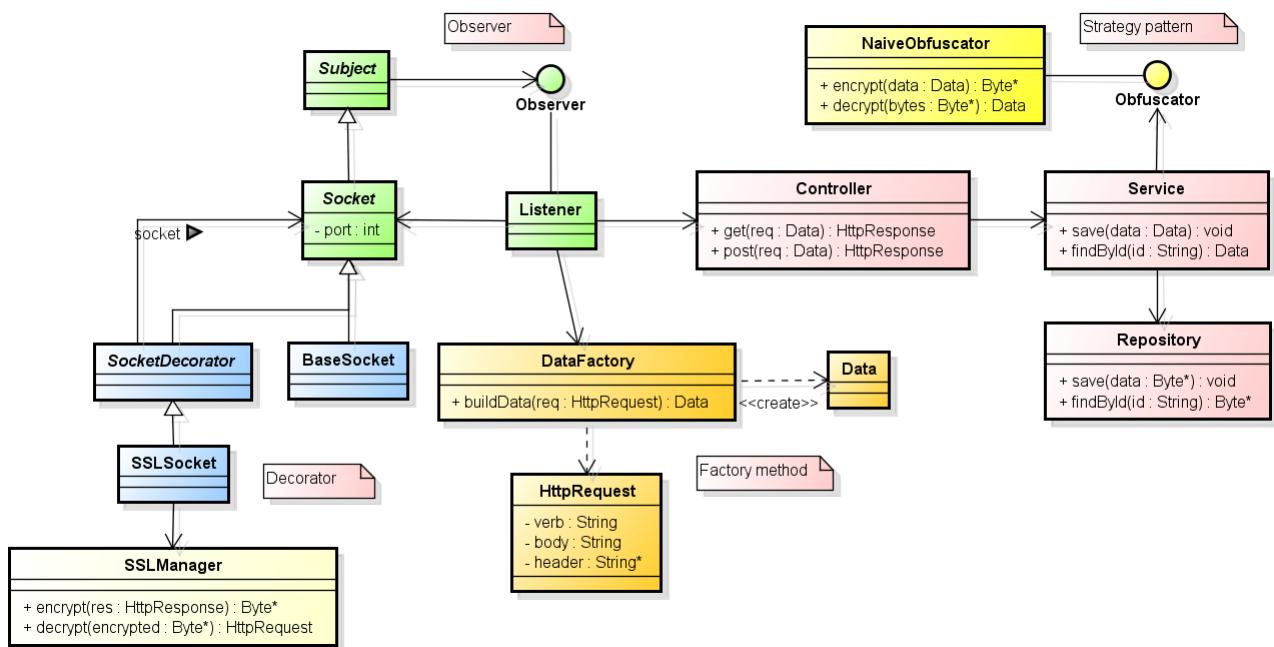
Descrizione

Una giovane *startup* italiana vuole creare un servizio web che consenta ad un qualsiasi utente di salvare una stringa associandola ad una chiave. Il salvataggio avviene nel cloud della *startup*. Il servizio deve esporre un'interfaccia REST che attraverso l'utilizzo di richieste HTTP con i verbi HTTP POST e GET permettano rispettivamente di salvare e recuperare successivamente la coppia (key, value). La struttura dati fornita con le richieste (HTTP body) è simile al seguente JSON { 'key': 'chiave', 'value': 'valore' }. Esso viene processato per creare una struttura dati opportuna che ne consente una più semplice elaborazione. Il servizio REST è implementato utilizzando un *listener* che resta in ascolto di connessioni sulla porta 80 nel caso di connessioni in HTTP. Esiste inoltre un'estensione del listener che utilizza la crittazione (HTTPS), restando in ascolto sulla porta 443. I dati crittati necessitano di essere ovviamente decifrati prima di essere elaborati ulteriormente. Le informazioni relative alla coppia (key, value) vengono infine salvate all'interno di un database in formato binario. Un algoritmo proprietario, ancora in via di miglioramento, comprime e decomprime le informazioni verso e dal database.

Si modelli il sistema descritto utilizzando un diagramma delle classi e gli opportuni *design pattern*. Inoltre, si descriva utilizzando un diagramma di sequenza una richiesta di lettura del valore associato alla chiave pippo.

Soluzione

La soluzione prevede l'utilizzo dei seguenti *design pattern*: Observer, Decorator, Factory Method e Strategy. La struttura è inoltre utilizzata una classica struttura a *tier* Controller – Service – Repository.



Esercizio 3 (3 punti)

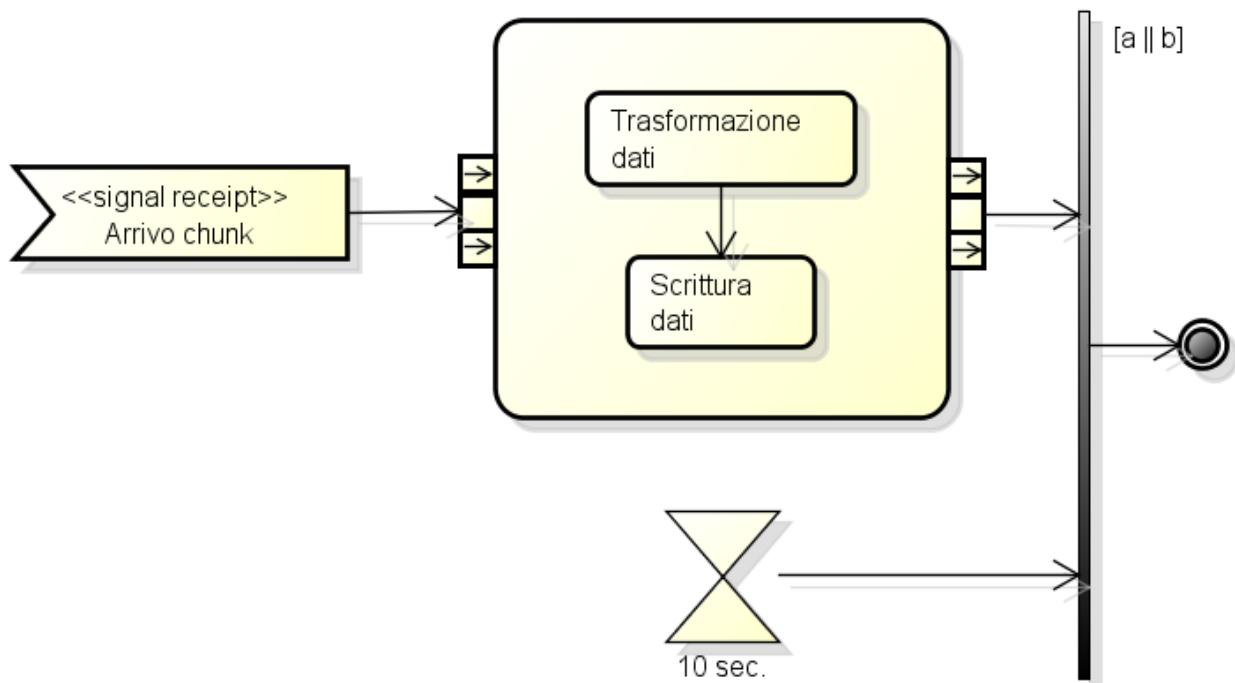
Descrizione

Apache Spark è una libreria che consente di effettuare analisi sui Big Data, ossia su volumi di dati considerevoli e solitamente non strutturati. La componente *streaming* della libreria può essere utilizzata per eseguire analisi su grandi moli di dati il cui arrivo è continuo. Spark Streaming utilizza la tecnica del *micro batching*: anziché elaborare i dati appena arrivati uno alla volta, li accumula in gruppi di n (*chunk*), elaborandoli successivamente con una sola esecuzione.

Utilizzando un diagramma di attività e focalizzandosi sull'arrivo di un singolo *chunk*, si modelli una sua elaborazione, che consista nella trasformazione e scrittura su database di questi dati. Si preveda inoltre un *timeout* di 10 secondi, che faccia terminare prematuramente l'elaborazione nel caso in cui non arrivi alcun *chunk* da elaborare.

Soluzione

La soluzione prevede l'utilizzo di una regione di espansione.



Ingegneria del Software A.A. 2016/2017

Esame 2017-06-27

Esercizio 1 (6 punti)

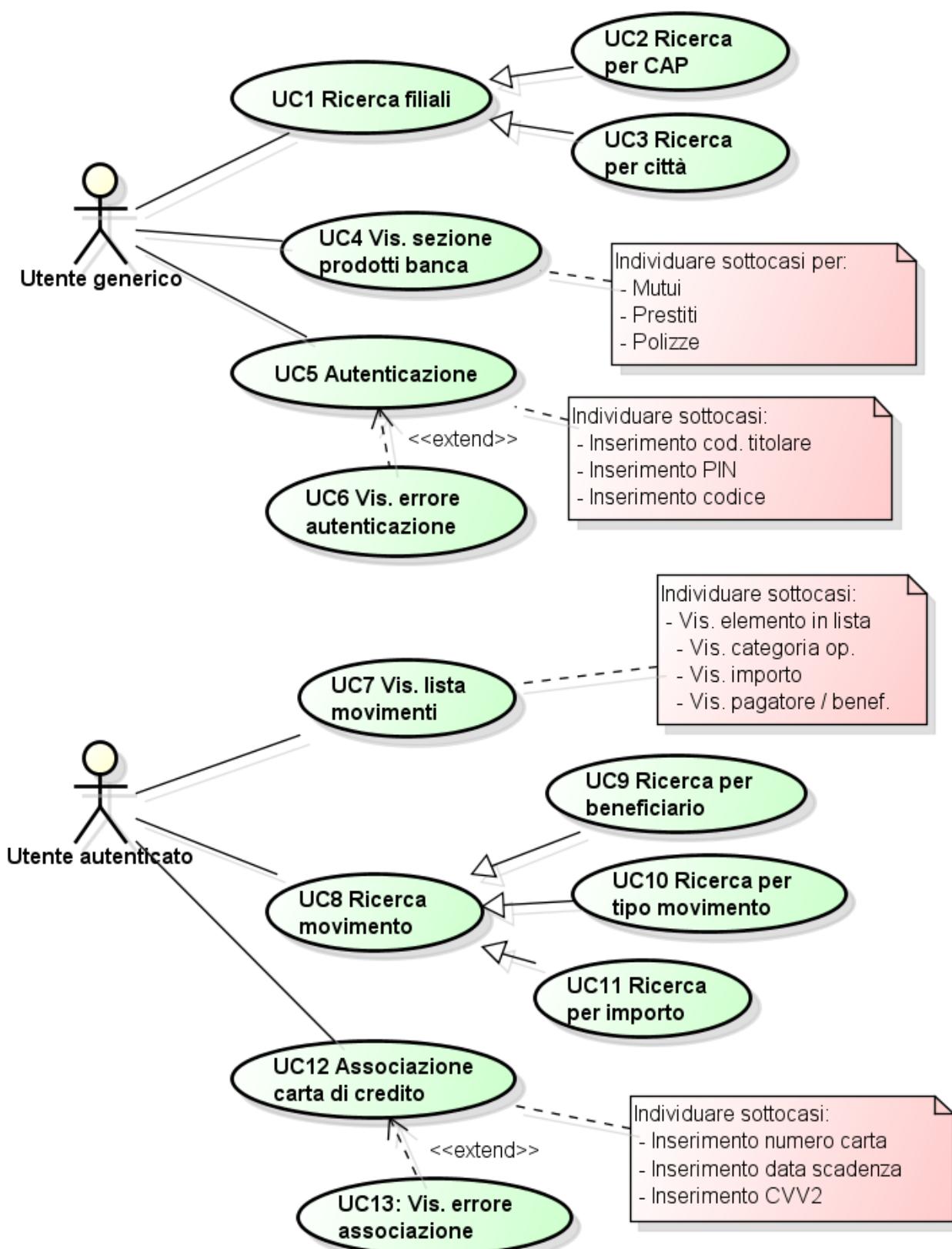
Descrizione

Il sito online di una banca fa parte dei progetti chiamati di “multicanalità”. Attraverso il sito è possibile sia recuperare delle informazioni generiche, sia effettuare delle operazioni dispositivo. In particolare, per essere riconosciuto dalla banca, un utente deve inserire il proprio codice titolare, un PIN ed un codice random, solitamente generato *ad hoc* da un token fisico. All'interno della pagina a lui dedicata, un utente può quindi vedere la lista di movimenti effettuati sul suo conto negli ultimi 30 giorni. Per ogni movimento, nella lista sono riportati la categoria di operazione (pagamento stipendio / pensione, operazione acquisto alimentari, pagamento F24, ...) e l'importo. Se il movimento è un prelievo, viene quindi visualizzato il beneficiario, altrimenti viene visualizzato il pagatore. Per facilitare l'utenza nel consultare i propri movimenti, sono disponibili nel sito diversi tipi di ricerca: per beneficiario, per tipologia di movimentazione e per importo. All'interno del sito, un utente può inoltre associare una carta di credito, inserendo il numero di carta, la data di scadenza e il codice CVV2. Un errore informa l'utente se le informazioni inserite non sono corrette. Il sito della banca offre ovviamente anche funzioni per l'utenza generica. E' infatti possibile ricercare le filiali della banca per CAP o Città. È inoltre disponibile una sezione nella quale vengono descritti i prodotti della banca, quali mutui, prestiti e polizze vita / danni.

Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari descritti. Non è richiesta la descrizione testuale dei casi d'uso individuati.

Soluzione

I sottocasi individuati come commenti devono essere sempre sviluppati durante la prova d'esame.



Esercizio 2 (7 punti)

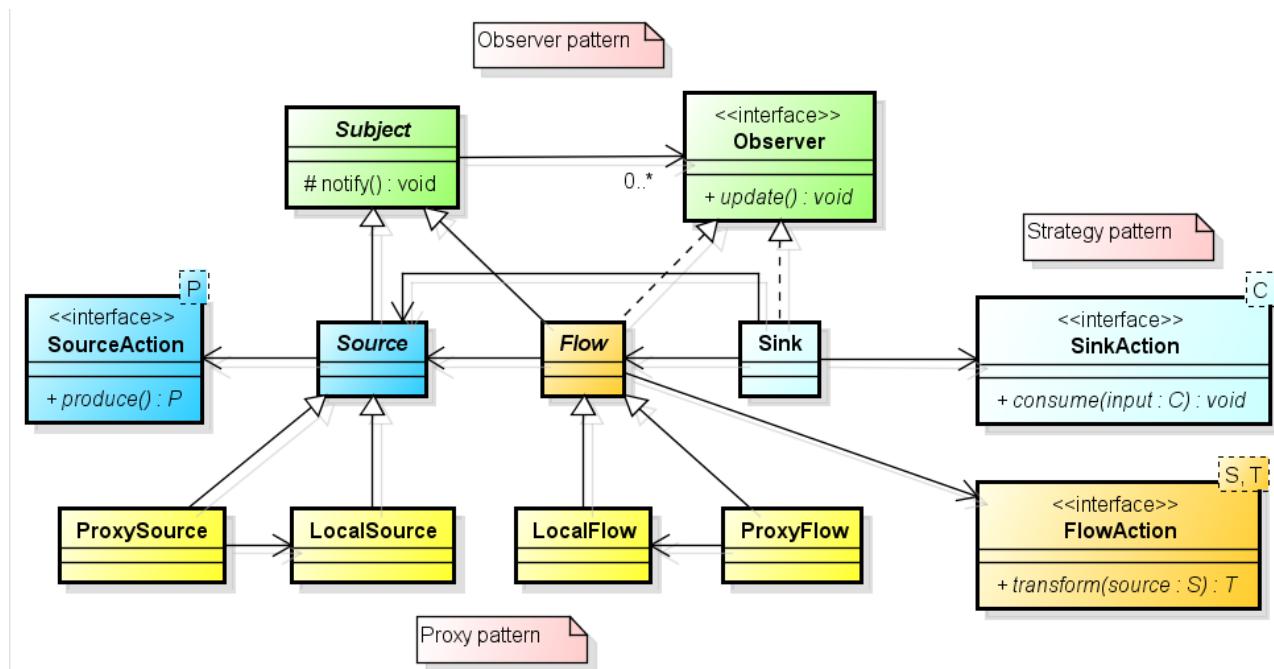
Descrizione

La programmazione moderna è sempre più orientata all'utilizzo di un modello ad eventi. Anziché controllare proattivamente se l'evento atteso sia disponibile, una risorsa viene informata di questo fatto. Questo tipo di programmazione è detta *reactive programming*. Le strutture che permettono di gestire gli eventi in modo appropriato vengono dette *stream*. Uno stream può essere di tre tipi differenti. Uno stream di tipo Source genera eventi. Uno stream di tipo Sink consuma eventi. Uno stream di tipo Flow consuma un evento e ne genera un altro. Anziché utilizzare un tipo dedicato per gli eventi, queste strutture sono generiche sui tipi. Quindi, un oggetto di tipo Source può essere osservato da oggetti di tipo Sink o Flow, mentre un oggetto di tipo Flow può essere osservato da oggetti di tipo Sink. Source e Flow possono ovviamente essere locali all'applicazione o remoti. Per mantere distinta la logica di controllo degli stream e l'effettiva implementazione della componente di business, ogni tipo di stream ha associato un tipo *Action. Abbiamo quindi SourceAction, FlowAction e SinkAction. Ognuno di questi tipi espone un unico opportuno metodo attraverso il quale è possibile implementare il comportamento dello stream.

Si modelli il sistema descritto utilizzando un diagramma delle classi e gli opportuni *design pattern*. Inoltre, si descriva utilizzando un diagramma di sequenza una catena di stream, che, leggendo un file di testo su un file system remoto, lo stampano su una console.

Soluzione

Design pattern utilizzati: Observer pattern, Proxy pattern, Strategy pattern.



Esercizio 3 (3 punti)

Descrizione

Siano prese in considerazione le seguenti classi, che permettono di copiare dei semplici caratteri inseriti da tastiera direttamente verso una stampante.

```

public class Copier {

    private KeyboardReader reader;

    private PrinterWriter writer;

    public Copier(KeyboardReader reader, PrinterWriter writer) {
        this.reader = reader;
        this.writer = writer;
    }

    public void copy(OutputDevice dev) {
        char c;

        while ((c = this.reader.read()) != EOF)
            if (dev == PRINTER)
                writer.write(c);
    }
}

public class KeyboardReader {

    public char read() {
        // Reads a character from the keyboard
    }
}

public class PrinterWriter {

    public void write(char c) {
        // Writes the character directly into the printer spool
    }
}

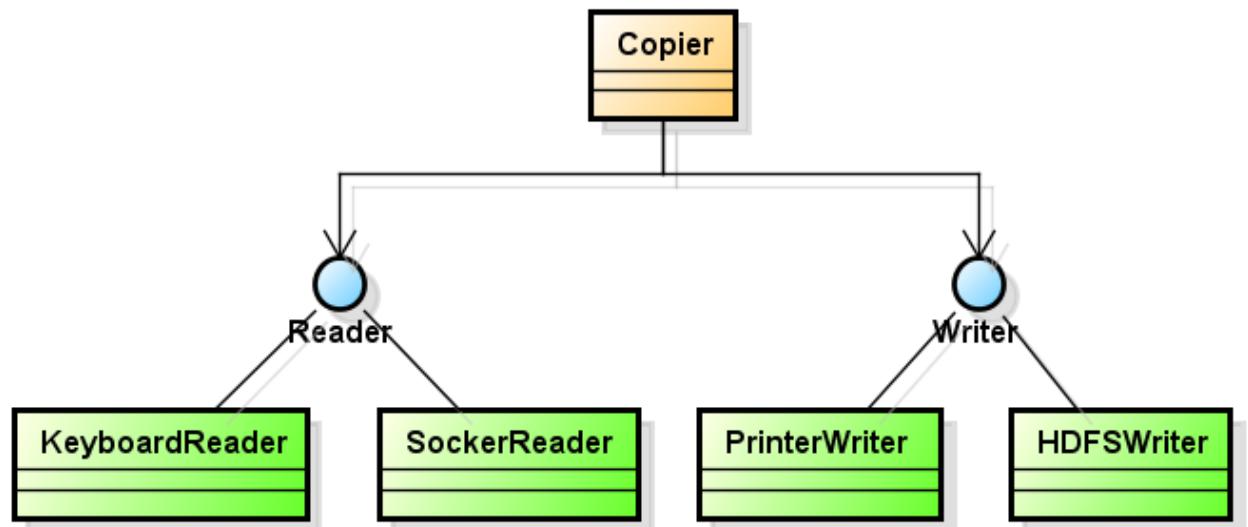
```

Si fornisca il diagramma delle classi di una soluzione che permetta di estendere le suddette classi in modo tale che la classe *Copy* possa essere riutilizzata per copiare caratteri da provenienti da un Socket direttamente su un file all'interno di un filesystem HDFS.

La soluzione dovrà tenere in considerazione il *Dependency Inversion Principle*.

Soluzione

Il livelli più alti dell'architettura non devono dipendere dalle implementazioni dei livelli più bassi. Copier ora dipendende solo da astrazioni, così come i *reader* e *writer* concreti.



Ingegneria del Software A.A. 2016/2017

Esame 2017-07-13

Esercizio 1 (6 punti)

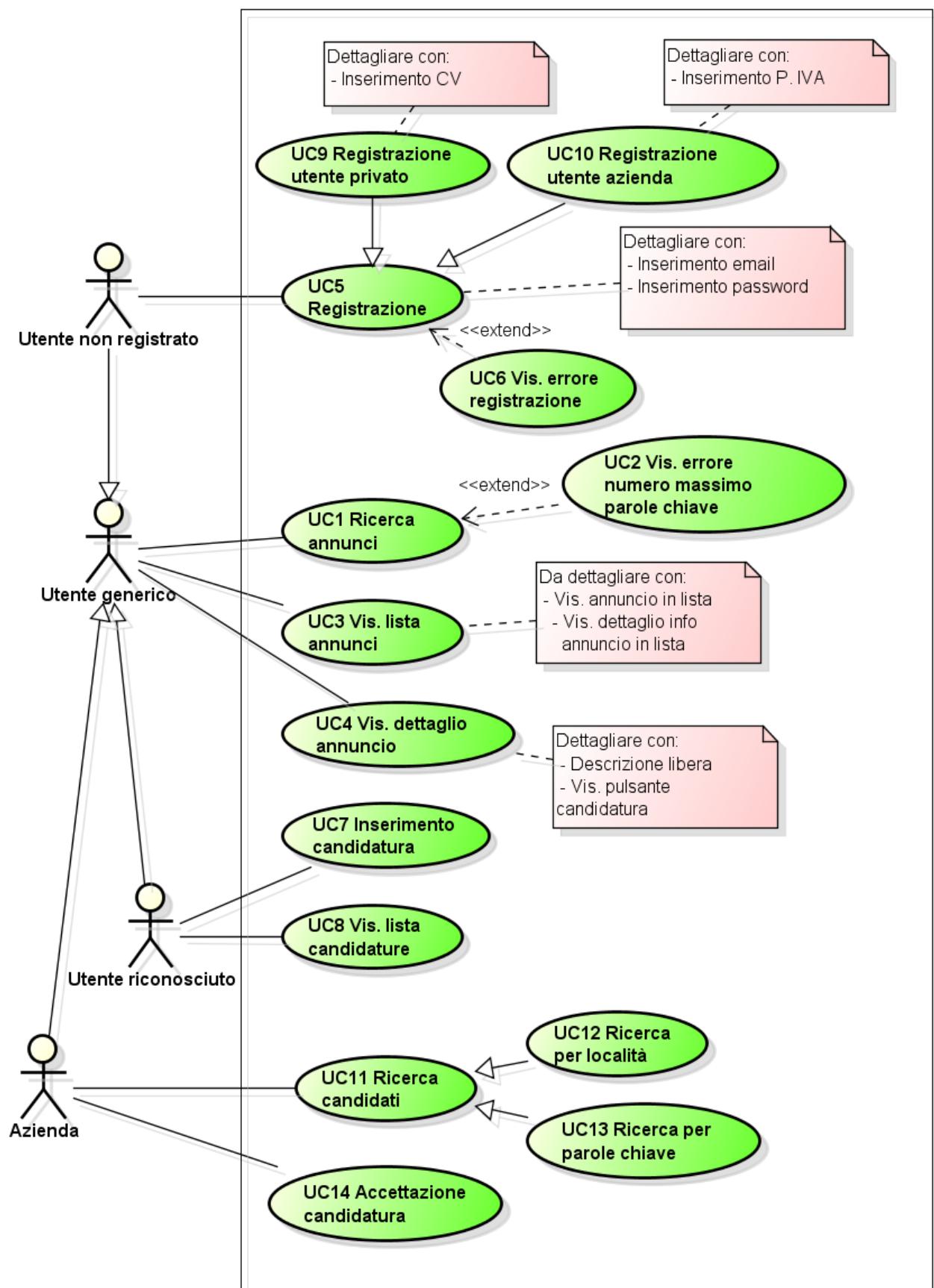
Descrizione

Monster è uno dei principali siti sul mondo del lavoro al mondo. Al suo interno chiunque può ricercare tra le offerte di lavoro esistenti. Le aziende, invece possono inserire le posizioni aperte che stanno ricercando. Qualsiasi utente può ricercare gli annunci disponibili semplicemente inserendo da una a cinque parole chiave o una località di riferimento. Il risultato di questa ricerca è una lista di annunci, che per ciascuno di essi riporta un titolo, l'azienda che propone l'annuncio e la posizione geografica del lavoro. Il dettaglio di un annuncio visualizza una descrizione libera più lunga e la possibilità di candidarsi. Per potersi candidare un utente deve essere registrato. La registrazione richiede l'inserimento di un'email valida, di una password e di un *curriculum vitae*. La candidatura invia un'email all'utente e la aggiunge alla sua lista delle candidature. Anche un'azienda per poter inserire annunci deve registrarsi, inserendo anche la partita iva. Un'azienda può ricercare gli altri utenti per località o per parole chiave. Infine, può accettare la candidatura di un utente ad un annuncio di lavoro. Il sistema invierà un'email per segnalare l'accettazione.

Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari descritti. Non è richiesta la descrizione testuale dei casi d'uso individuati.

Soluzione

Dove è riportato il commento “Da dettagliare” devono essere individuati obbligatoriamente i sottocasi d'uso riportati nel testo.



Esercizio 2 (7 punti)

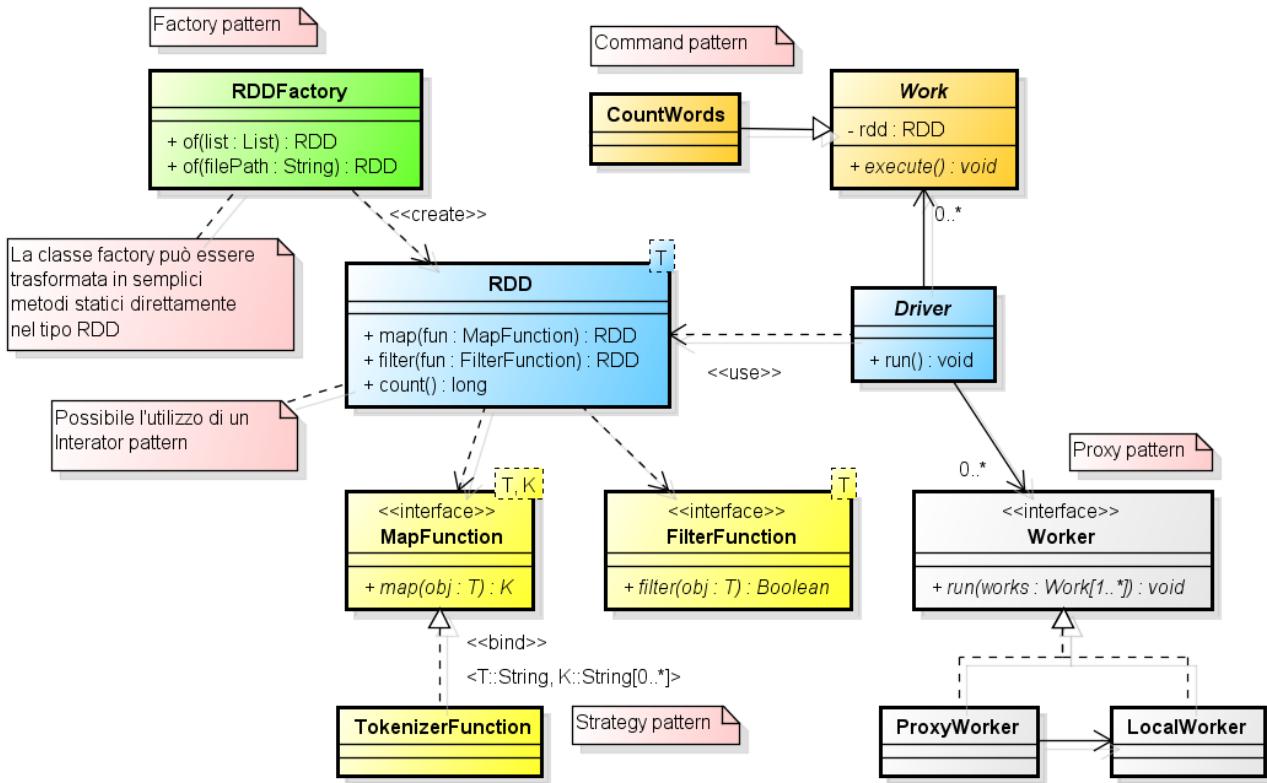
Descrizione

Apache Spark è un motore di esecuzione per processi che elaborano grandi quantità di dati. Il suo punto di forza principale è quello di essere in grado di parallelizzare l'esecuzione di algoritmi in modo distribuito, pur mantenendo un'API semplice e per lo più tipo sequenziale. Il fulcro principale è il tipo `RDD[T]`. Questa struttura può essere pensata tranquillamente come ad una lista di elementi di tipo `T`, ma distribuita fra più macchine, dette *worker*. La lista viene però dichiarata su una macchina principale, detta *driver*, che è anche quella che esegue il programma principale. La costruzione di un `RDD` avviene utilizzando opportuni metodi, che ne creano una nuova a partire da diverse sorgenti: ad esempio da un oggetto di tipo `List[T]` o da un file creando un `RDD[String]`. La lista espone operazioni come `map` e `filter`. Entrambi questi metodi hanno in input un algoritmo che lavora su un elemento di tipo `T`. La differenza è che `map` trasforma un oggetto di tipo `T` in un oggetto di tipo `K`, mentre `filter` dato un oggetto di tipo `T`, ritorna un valore booleano vero o falso. Il metodo `run` del programma *driver*, quindi, si occupa di inviare le istruzioni da eseguire su una porzione di `RDD` ad ogni *worker*. Poiché il programma da eseguire non ha una struttura fissa, è necessario utilizzare una struttura che ne permetta l'esecuzione in modo uniforme. Si ricorda che l'esecuzione sui *worker* avviene in modo remoto rispetto al *driver*.

Si modelli il sistema descritto utilizzando un diagramma delle classi e gli opportuni *design pattern*. Inoltre, si descriva utilizzando un diagramma di sequenza un programma che conta quante parole sono presenti all'interno di un file di testo. Il file verrà dapprima letto riga per riga, poi suddiviso in parole, che alla fine saranno conteggiate con un opportuno metodo `count`.

Soluzione

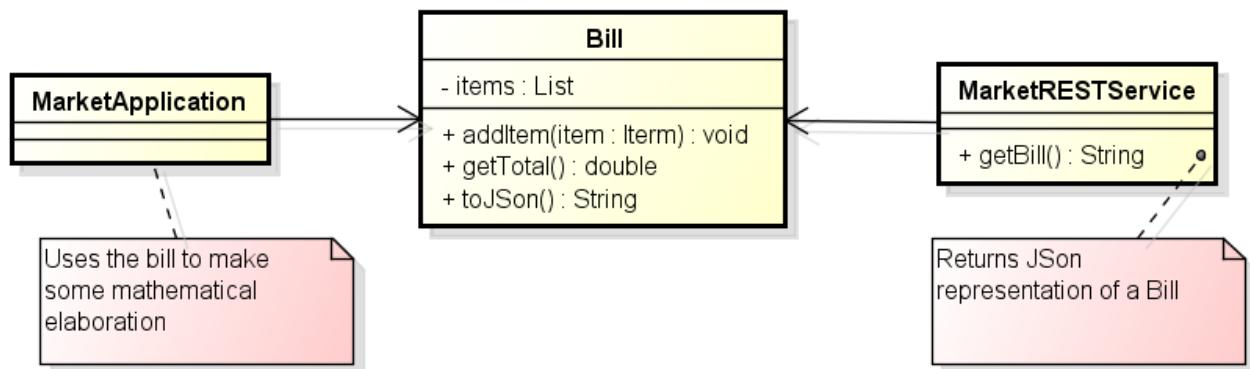
La soluzione prevede l'utilizzo dei seguenti design pattern: Command pattern, Proxy pattern, Strategy pattern, Factory Method pattern.



Esercizio 3 (3 punti)

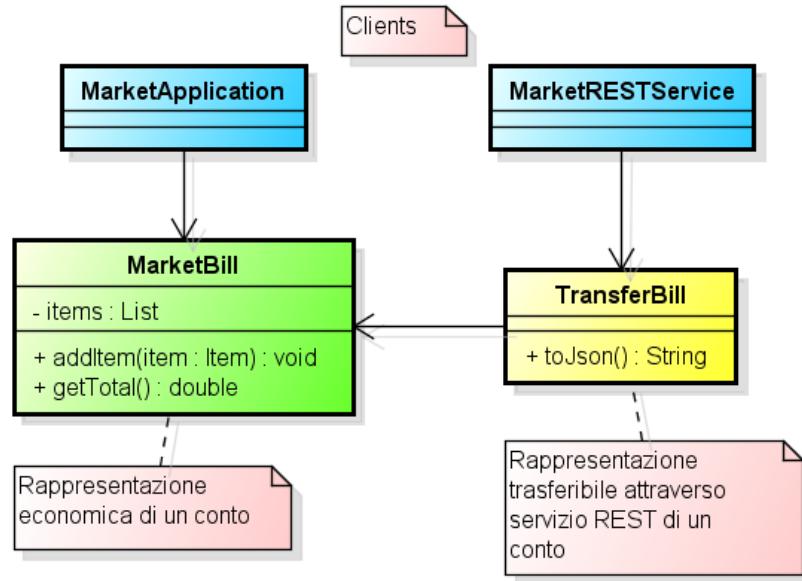
Descrizione

La classe **Bill** è utilizzata per gestire oggetti che rappresentano liste della spesa sia dal punto di vista del calcolo matematico, sia da una applicazione che espone queste informazioni in formato Json tramite un servizio REST. Dato il seguente diagramma delle classi, se ne fornisca una versione modificata in modo tale che la soluzione aderisca al “Single Responsibility Principle”.



Soluzione

Le funzionalità del tipo **Bill** devono essere suddivise in opportuni tipi, in modo da suddividere gli assi di cambiamento. In questo modo, se la rappresentazione Json di un conto dovesse cambiare, **MarketBill** non verrebbe modificato. D'altra parte, se l'algoritmo di calcolo del totale dovesse cambiare, **TransferBill** non verrebbe modificato.



Ingegneria del Software A.A. 2016/2017

Esame 2017-08-29

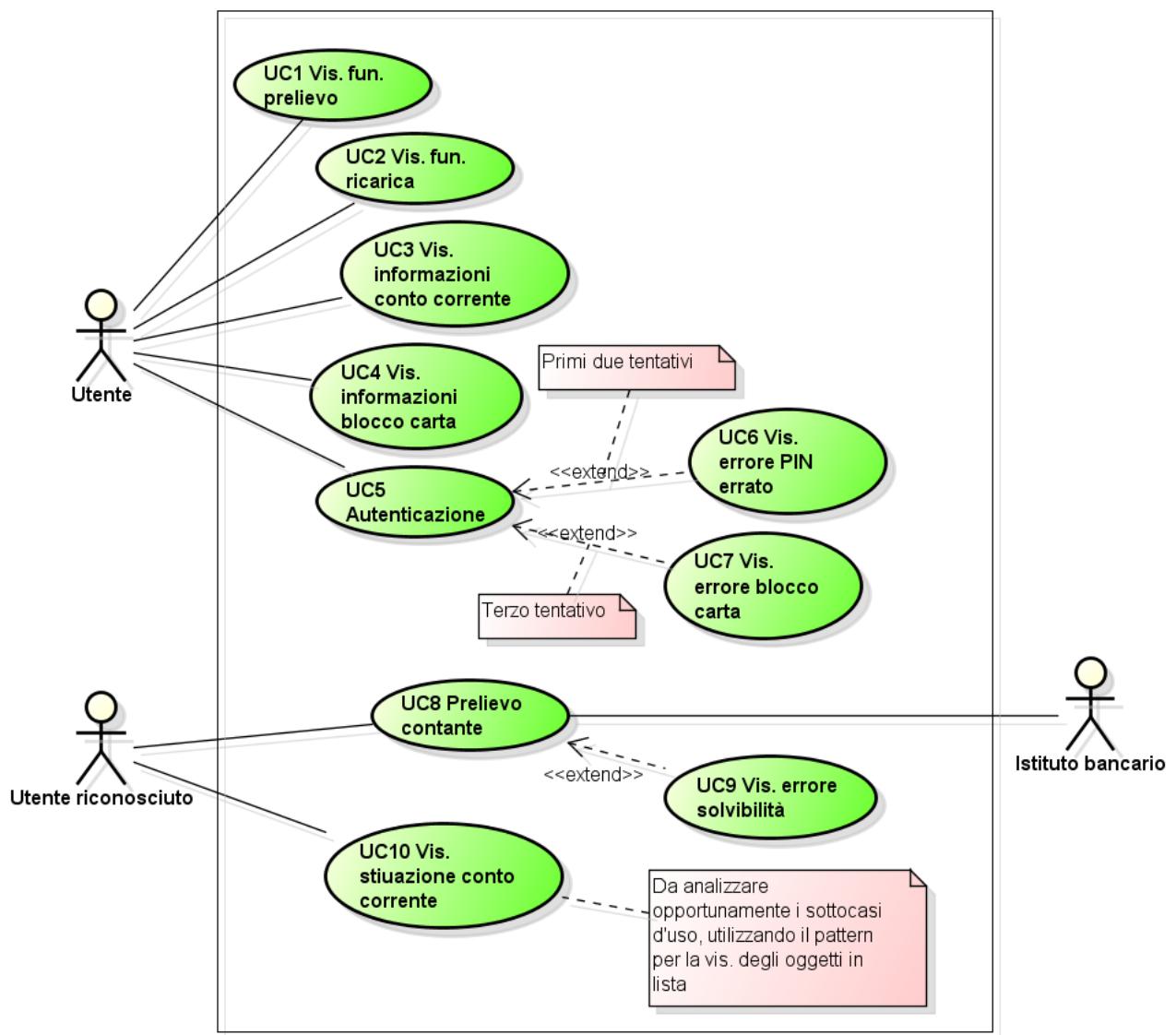
Esercizio 1 (6 punti)

Descrizione

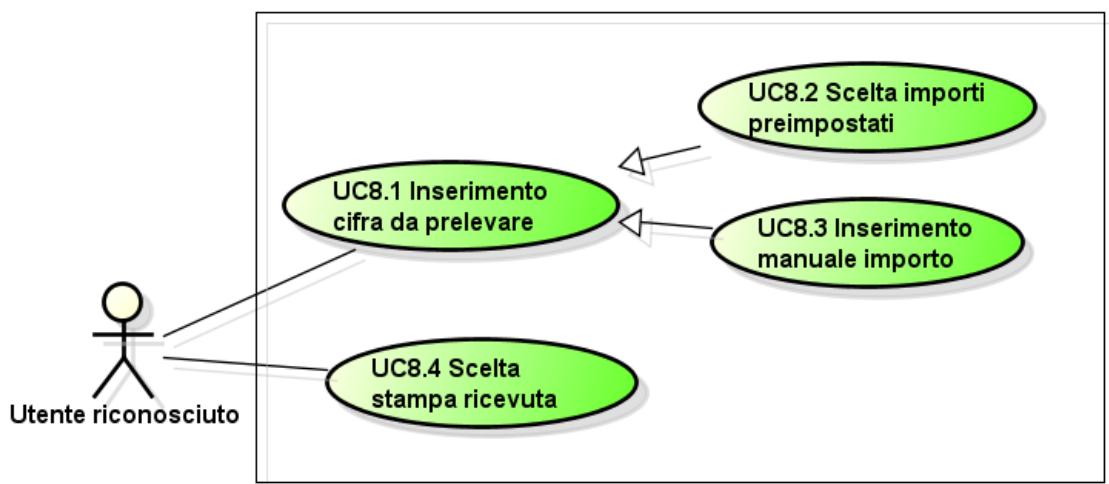
Lo sportello ATM (“bancomat”) può essere visto come una semplice interfaccia esposta verso un utente. Inizialmente, essa presenta funzionalità quali il prelievo, la ricarica telefonica, la visualizzazione della situazione del conto corrente aperto presso quell’istituto, e le istruzioni per bloccare la propria carta di debito/credito in caso di furto o smarrimento. L’accesso a tali funzionalità richiede l’inserimento di un PIN di 5 cifre. Fino a due errori di immissione del PIN in una sessione di utilizzo, l’interfaccia visualizza un messaggio di errore. Al terzo errore, il sistema blocca la tessera bancomat dell’utente, visualizzando la notifica corrispondente. Il prelievo di contante richiede all’utente di specificare la cifra da erogare: è possibile scegliere tra proposte preimpostate oppure inserire un altro importo usando un apposito tastierino. L’ATM dialoga poi con l’istituto bancario di appartenenza dell’utente per effettuare verifiche di solvibilità dell’importo. In caso di risposta negativa, l’interfaccia ATM visualizza un opportuno messaggio di errore. Contestualmente al prelievo, l’utente può richiedere la stampa di una ricevuta. La funzionalità di visualizzazione della situazione del proprio conto corrente, mostra a video gli ultimi 10 movimenti effettuati su di esso. Per ciascun movimento in lista, la funzionalità visualizza l’importo, il beneficiario (per le uscite) o l’ordinante (per le entrate), e la data di esecuzione del versamento. Si utilizzino i diagrammi dei casi d’uso per modellare gli scenari descritti. Non è richiesta la descrizione testuale dei casi d’uso individuati.

Soluzione

I casi d’uso indicati come “da analizzare” devono essere opportunamente riportati su un diagramma dei casi d’uso.



Il caso d'uso UC8 individua i seguenti sotto-casi d'uso.



Esercizio 2 (7 punti)

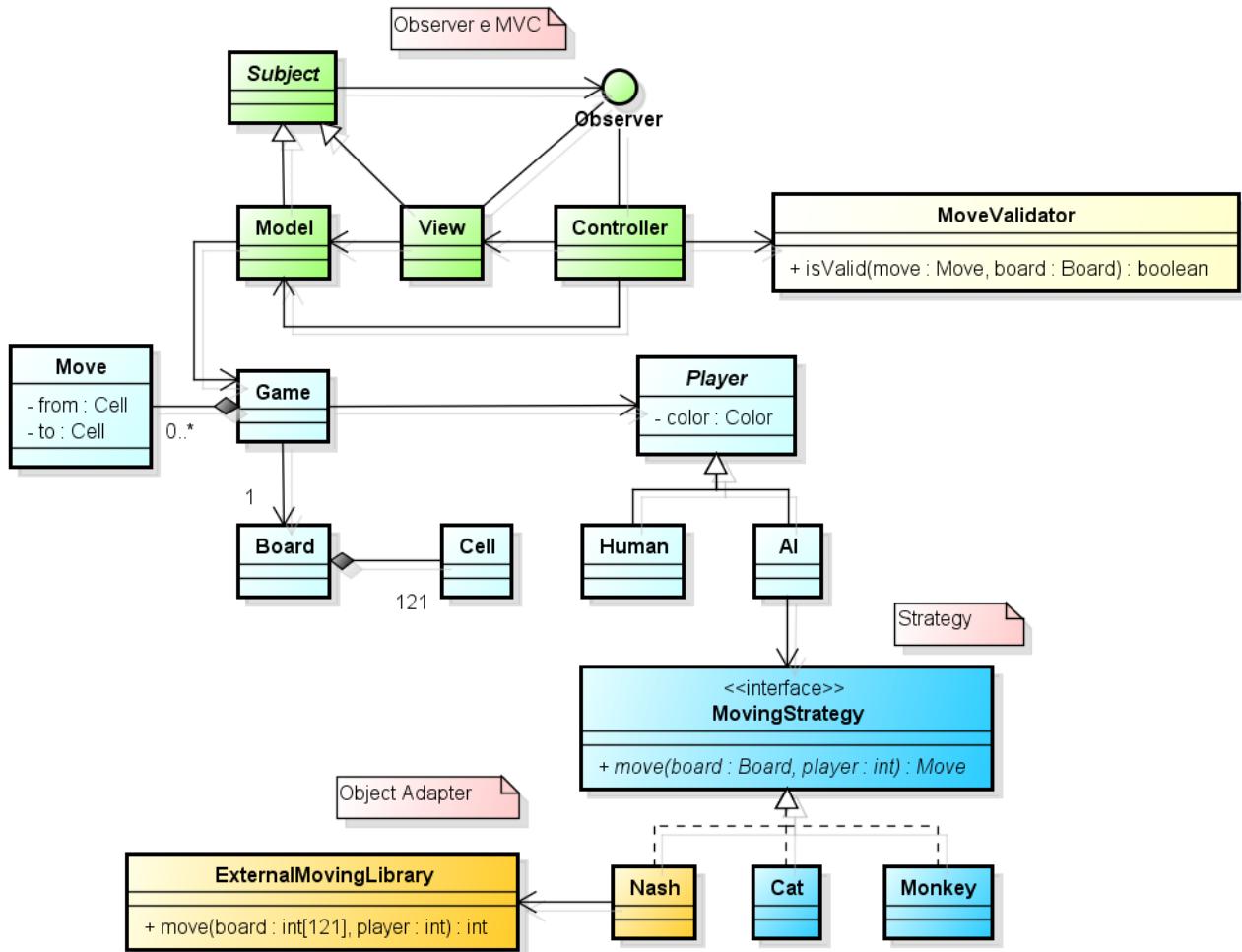
Descrizione

Hex è un gioco su scacchiera con celle esagonali re-inventato dal famoso matematico John F. Nash quando era studente a Princeton. La scacchiera ha forma di rombo con 11 celle per lato e pedine di due colori, uno per giocatore. All'inizio del gioco, ciascuno dei due lati opposti del rombo contiene pedine dello stesso colore, in modo tale che ogni giocatore occupi lati non contigui del rombo. Durante il gioco, ciascun giocatore, a turno, posiziona le proprie pedine sulle scacchiera, cercando di costruire un percorso continuo che unisca i due lati del proprio colore. Si immagini di dover progettare questo sistema di gioco, fornendo all'utente la possibilità di giocare contro un altro umano o contro una intelligenza artificiale (AI) di tipologia *monkey*, *cat*, *nash*, corrispondenti ad abilità crescenti. Tali tipologie di AI utilizzano una rappresentazione della scacchiera a matrice di celle, di tipo Cell. Per implementare il livello *nash* si desidera utilizzare una libreria esterna pre-esistente che espone un metodo `move(board: int[121], player: int): int`, che data una scacchiera linearizzata e l'indicazione del giocatore che deve muovere, ritorna la cella in cui porre la pedina. Ogni mossa deve essere validata opportunamente, in modo da garantire il rispetto delle regole di gioco. Per gli utenti umani, il sistema deve disporre di una interfaccia grafica che permetta loro di scegliere visivamente le proprie mosse.

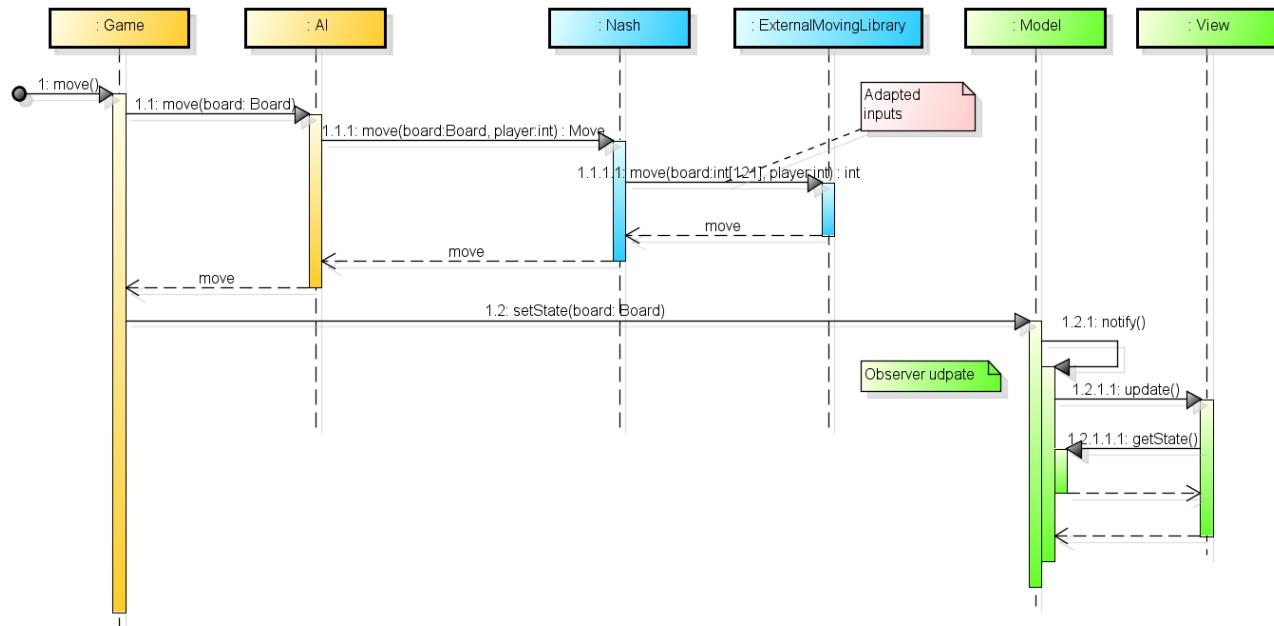
Si modelli il sistema descritto mediante un diagramma delle classi che utilizzi opportuni design pattern. Mediante un diagramma di sequenza, si descrivano poi le azioni corrispondenti alla mossa da parte di un'AI di tipo *nash*, fino al posizionamento della pedina scelta sulla scacchiera.

Soluzione

La soluzione prevede l'utilizzo dei seguenti design pattern: Observer, MVC, Strategy e Object Adapter.



Un esempio di diagramma di sequenza che modella le invocazioni necessarie per completare una mossa proveniente dall'AI di tipo Nash il seguente.



Esercizio 3 (3 punti)

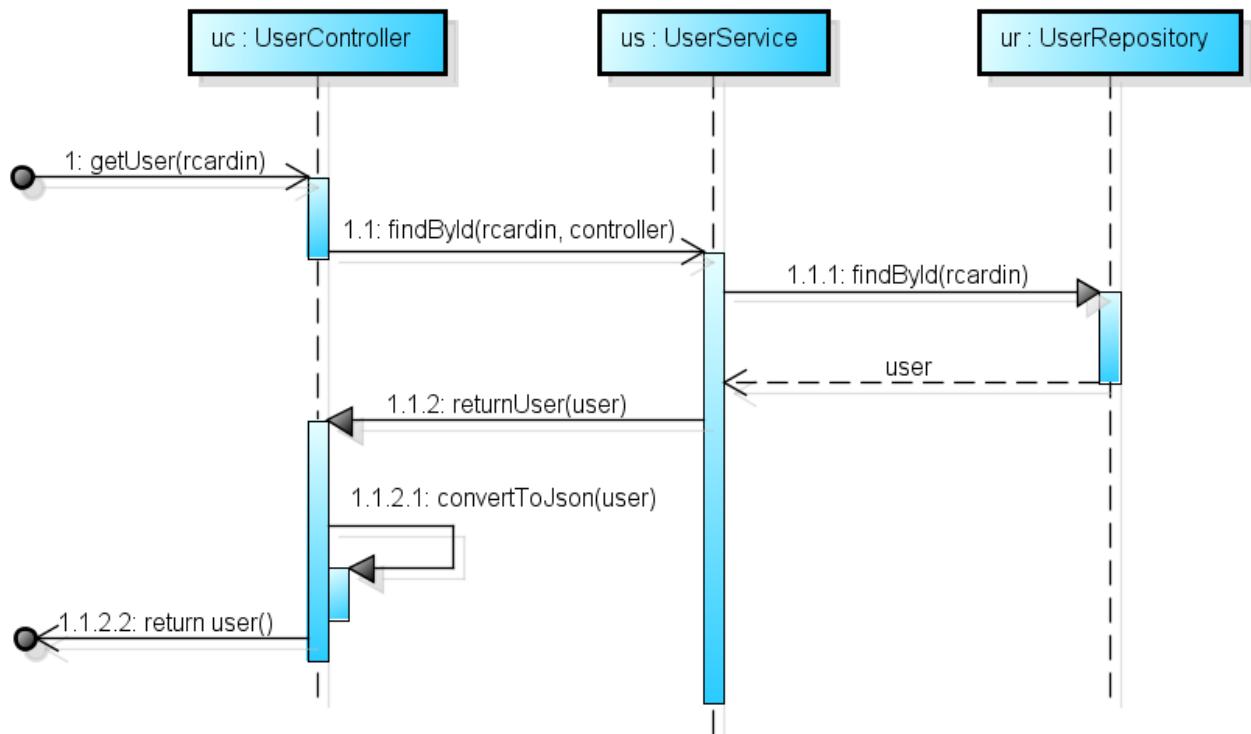
Descrizione

L'utilizzo del paradigma di programmazione reactive punta a ridurre il numero di operazioni sincrone (bloccanti) utilizzate, ritenendo che l'impiego di operazioni asincrone sfrutta meglio (senza attese) le risorse di calcolo a disposizione del programma. Una forma basica di questo paradigma utilizza funzioni di callback, come, per esempio, nel framework *Node.js*. Si assuma un programma reactive contenente una classe `UserRepository`, che esponga il metodo `findById(id: String): User`, che recupera da una base dati l'utente contraddistinto da un particolare id, e una classe `UserController`, che esponga il metodo `getUser(id: String): String`, che ritorna una rappresentazione in formato JSON di un utente con identificativo id.

Utilizzando un diagramma di sequenza, si modelli il recupero del JSON associato all'utente con id "rardin", implementando un approccio che preveda l'invocazione di almeno una funzione di callback. Si ricordi che, in un linguaggio a oggetti, essa può essere rappresentata con un pattern Strategy, ossia come implementazione di una functional interface, cioè un'interfaccia che espone un solo metodo. È possibile aggiungere ulteriori tipi al diagramma e/o modificare le firme dei metodi esistenti.

Soluzione

Una delle possibili soluzioni prevede l'utilizzo di un ulteriore livello di indirezione, ossia il tipo `UserService`. Si noti la natura asincrona dei messaggi tra il tipo controller e il tipo service.



Ingegneria del Software A.A. 2016/2017

Esame 2017-09-11

Esercizio 1 (6 punti)

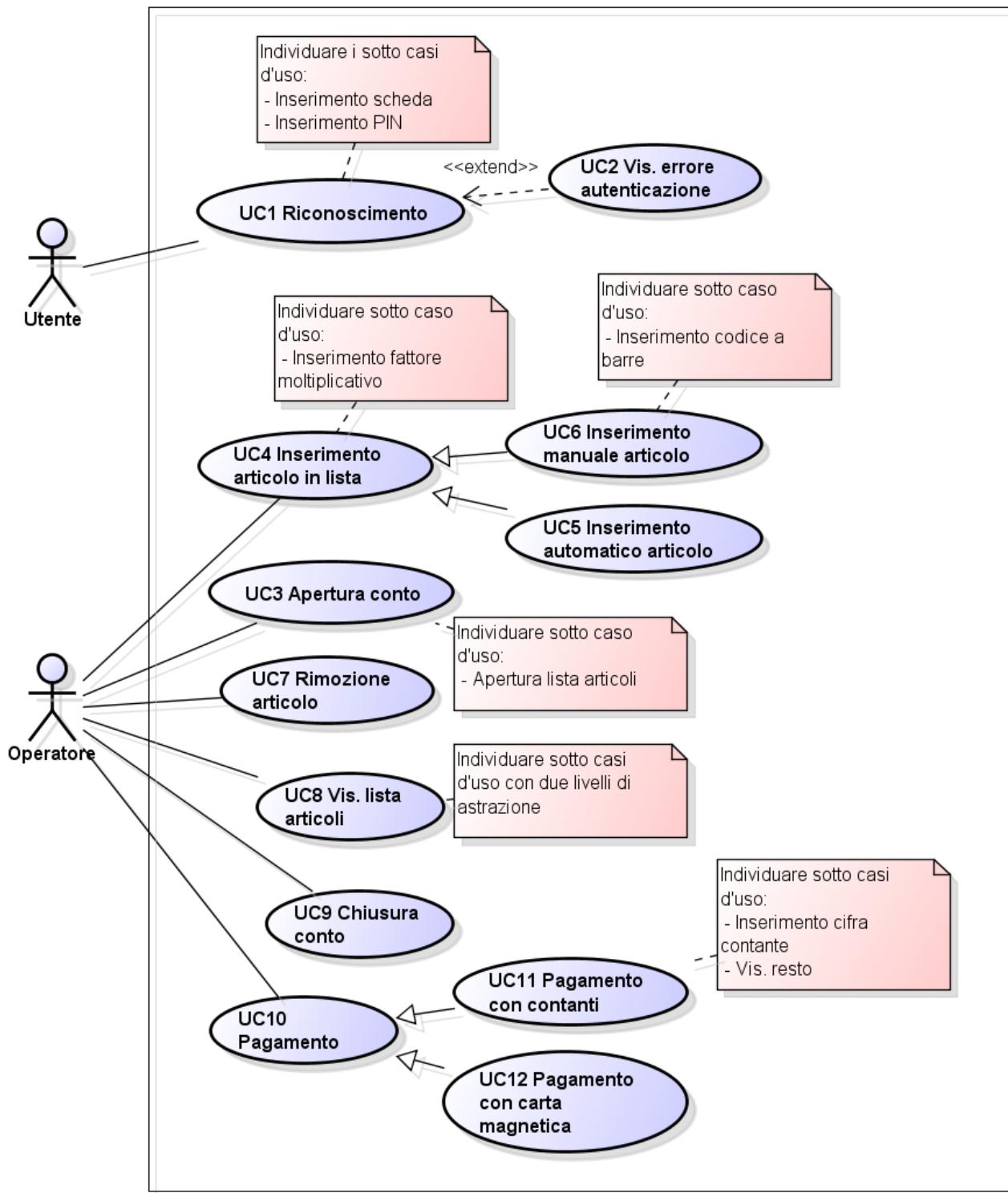
Descrizione

I programmi software installati nei registratori di cassa elettronici diventano sempre più sofisticati con l'evolvere della tecnologia. Per poter accedere alle funzionalità offerte da un regista di cassa moderno, l'operatore deve farsi riconoscere utilizzando una scheda magnetica, che riporta – tra l'altro – la username dell'operatore. Questi deve poi inserire a tastiera un PIN di 5 cifre che vale come password. L'errore di l'inserimento password causa la visualizzazione di un opportuno messaggio. Una volta riconosciuto, l'operatore può accedere a numerose funzionalità. Innanzitutto, può iniziare un nuovo conto per un cliente. Questa operazione consente di iniziare a creare la lista di articoli comprati, passando il codice a barre del prodotto sull'apposito lettore. Questa operazione contatta il sistema centralizzato del negozio che, accedendo a un DB, ritorna le informazioni di dettaglio del prodotto. È possibile rimuovere un articolo dalla lista del conto o inserirne uno digitando manualmente il codice a barre. Per l'acquisto di più unità dello stesso prodotto, è possibile specificare un fattore moltiplicativo intero da applicare al prezzo unitario. A fine conto, è possibile visualizzare la lista degli articoli inseriti. Tale lista, per ogni articolo riporta le seguenti informazioni: numero, breve descrizione e prezzo complessivo della riga. Il pagamento di un conto finale può avere tramite contanti, carta di credito o debito. Nel primo caso, l'operatore inserisce l'importo del contante ricevuto dal cliente e riceve in risposta l'eventuale resto.

Si utilizzino diagrammi dei casi d'uso per modellare gli scenari sopra descritti. Non ne è richiesta la descrizione testuale.

Soluzione

Individuare e modellare esplicitamente i sotto casi d'uso ove richiesto.



Esercizio 2 (7 punti)

Descrizione

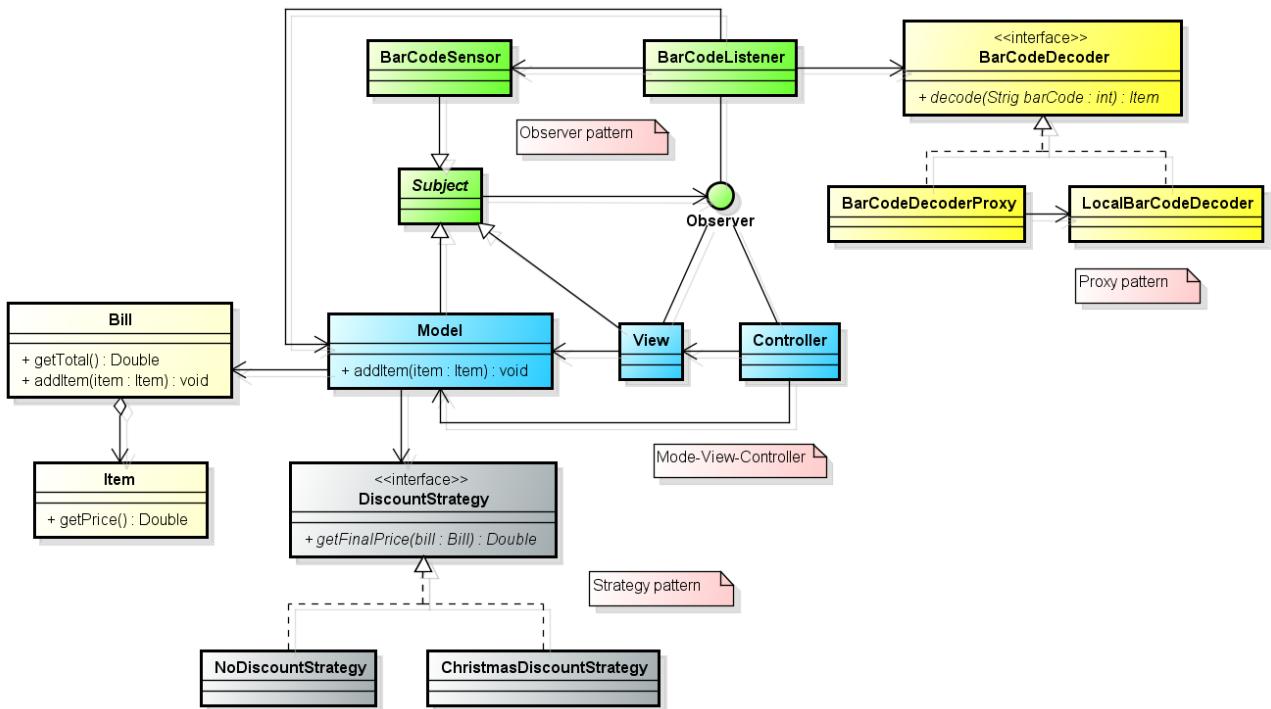
Il software installato in un registratore di cassa come quello descritto nell'esercizio 1, è abbastanza complesso da dover essere progettato utilizzando best practice architetturali. Oltre al software che gestisce l'interfaccia utente, il registratore di cassa possiede altri componenti. Il lettore di codice a barre resta in attesa del passaggio di un prodotto da parte dell'operatore. La decodifica del codice a barre viene svolta da

un sistema centralizzato remoto, che opera come se fosse locale al registratore di cassa. Le informazioni che esso ritorna sono racchiuse all'interno del tipo `Item`. Il conto è rappresentato dal tipo `Bill`, che è un aggregato di oggetti di tipo `Item`. Il calcolo del totale avviene utilizzando le informazioni contenute nel conto, applicandovi eventuali sconti, calcolati direttamente dal registratore di cassa. Gli sconti vengono applicati utilizzando diverse strategie, a seconda della campagna di marketing attualmente in atto.

Si modelli il sistema sopra descritto, mediante un diagramma delle classi che includa opportuni design pattern. Utilizzando poi un diagramma di sequenza, si descrivano le azioni corrispondenti alla lettura del codice a barre di un prodotto e alla visualizzazione a video delle sue informazioni all'interno del conto aperto.

Soluzione

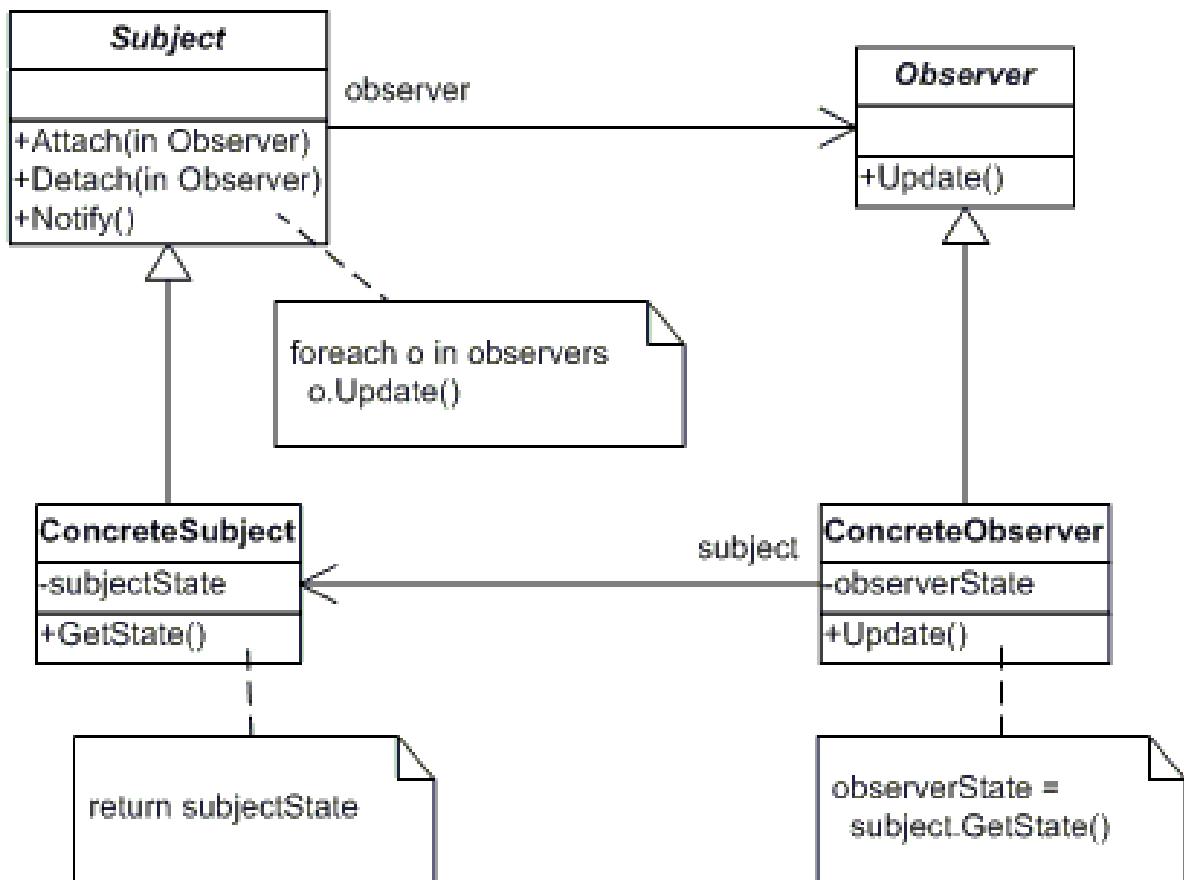
I pattern da applicare sono i seguenti: MVC, Observer, Proxy e Strategy pattern. Eventualmente è possibile individuare un Iterator pattern sul tipo `Bill`.



Esercizio 3 (3 punti)

Descrizione

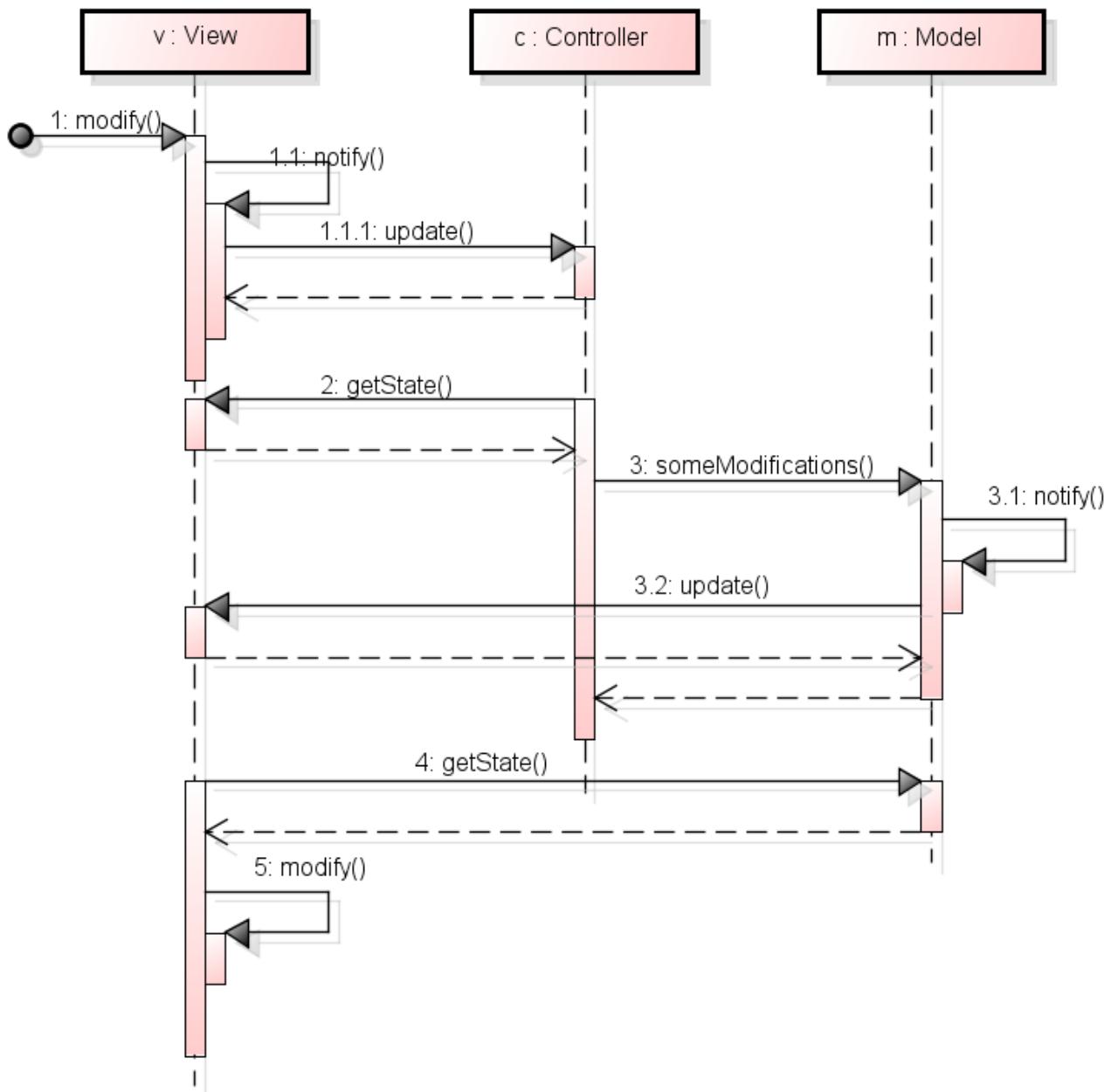
Nella sua versione base, nota come push model, il pattern Model View Controller, MVC, viene implementato utilizzando una doppia istanza del pattern Observer. Il diagramma delle classi qui di seguito rappresenta il pattern Observer.



Utilizzando un diagramma di sequenza, si modelli lo scambio di messaggi fra le componenti del pattern MVC che corrisponde al seguente flusso di operazioni: l'utente effettua un'operazione sulla vista, che modifica il modello e, di conseguenza, le informazioni visualizzate nella vista stessa.

Soluzione

Segue il diagramma di sequenza corrispondente.



Ingegneria del Software A.A. 2017/2018

Esame 2018-04-23

Esercizio 1 (6 punti)

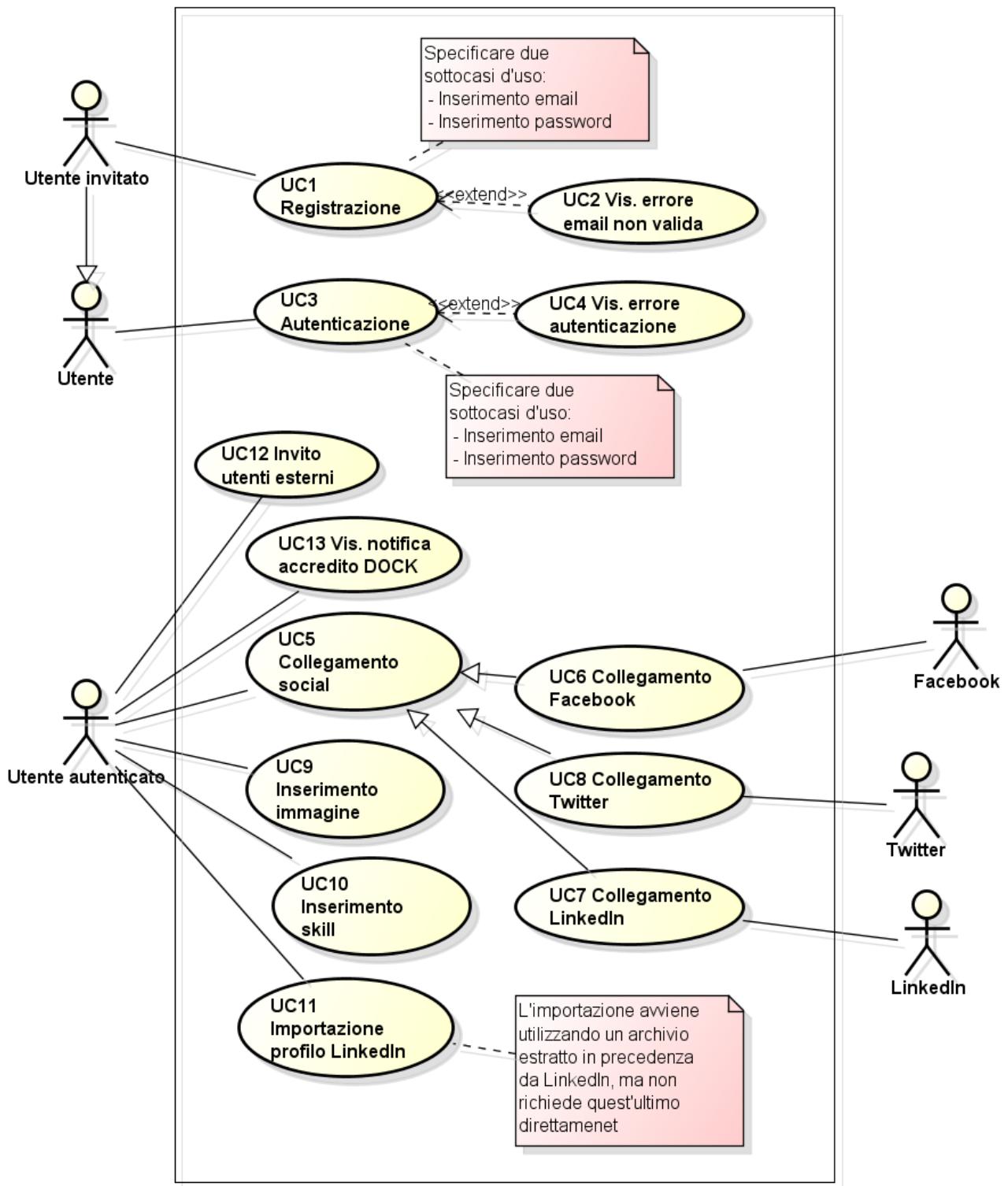
Descrizione

Dock.io è un nuovo progetto che prevede l'utilizzo di una Blockchain per condividere e validare le esperienze social. Per ora la registrazione al sito è possibile unicamente tramite invito e richiede l'inserimento di un'email valida e di una password. Le medesime informazioni sono richieste in fase di autenticazione. Per validare il proprio profilo è necessario collegare almeno uno dei seguenti social: LinkedIn, Facebook o Twitter. Inoltre è necessario inserire un'immagine del proprio profilo ed inserire almeno 5 *skill*, con indicazione del livello di esperienza associato: junior, middle e senior. L'inserimento di queste informazioni rende il profilo "verificato". E' inoltre possibile importare le proprie informazioni dal proprio profilo LinkedIn, utilizzando un archivio compresso precedentemente esportato dal social network esterno. Infine, è possibile invitare altre persone ad iscriversi a Dock.io. Per ogni nuovo account invitato, che completa l'iscrizione fino allo stato di profilo "verificato", vengono accreditati 44 DOCK, un *crypto token* appositamente creato. L'informazione viene notificata all'utente con un apposito messaggio interno.

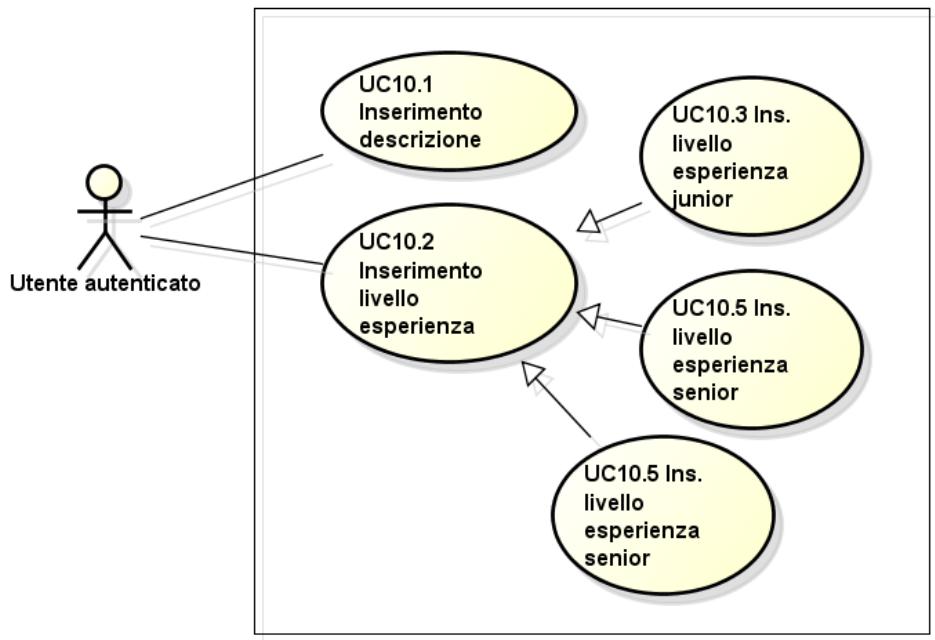
Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari descritti. Non è richiesta la descrizione testuale dei casi d'uso individuati.

Soluzione

Una possibile soluzione per il diagramma principale è la seguente.



Inserimento di una skill può essere dettagliato utilizzando il seguente diagramma.



Esercizio 2 (7 punti)

Amazon distribuisce i propri libri sui dispositivi Kindle utilizzando una metodologia *publish-subscribe*. Per ogni utente, Amazon mantiene una libreria dedicata nei propri server. Ogni qualvolta un utente compra un libro in formato Kindle, tutti i suoi dispositivi vengono notificati e, se disponibile una connessione, scaricano il nuovo libro. Per ottimizzare la memoria utilizzata, il libro non viene caricato tutto in memoria, ma oggetti pesanti come le immagini vengono caricate unicamente quando visualizzate a schermo. Ne deriva che ogni pagina del libro è renderizzata *just-in-time*, ossia solo all'occorrenza. Per questo un libro può essere visto come una collezione di pagine, dove l'algoritmo di attraversamento è particolare per ogni tipologia di Kindle. Il tutto è implementato utilizzando un pattern MVC sul dispositivo.

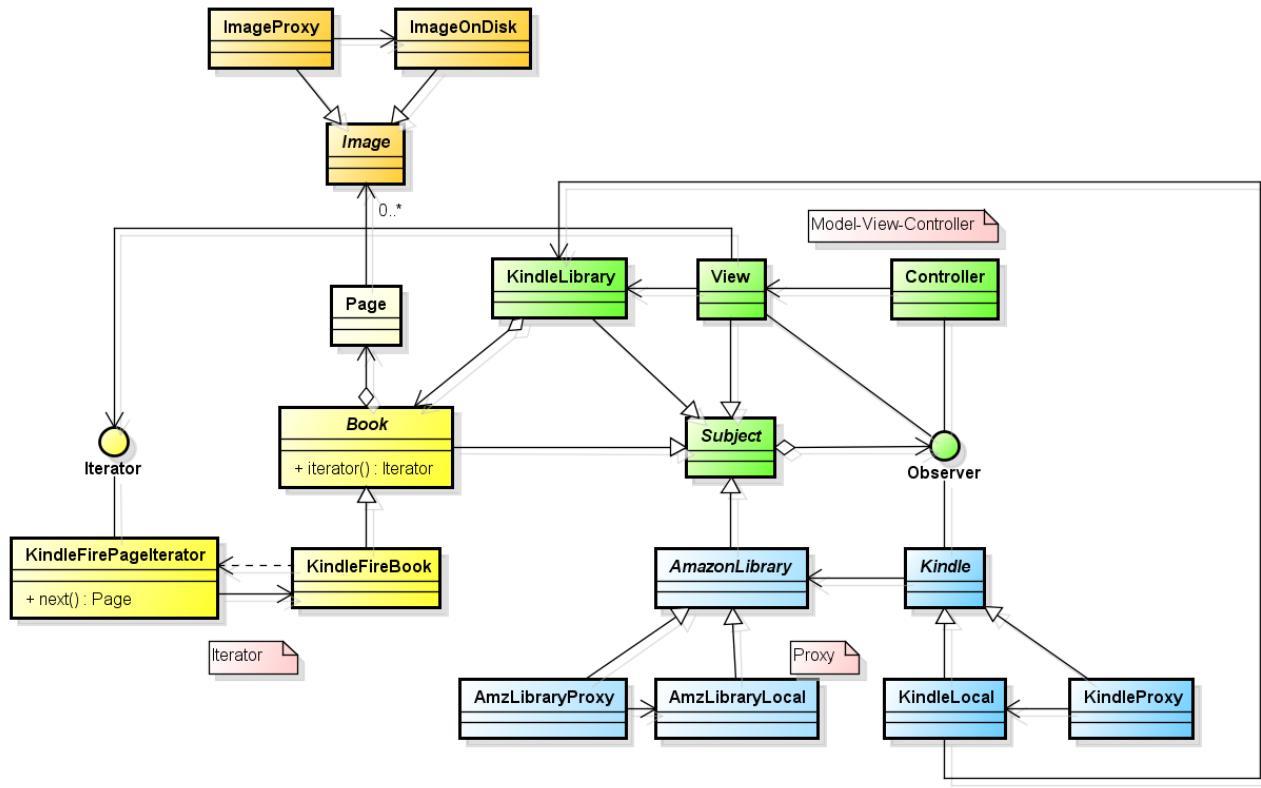
Si modelli il sistema descritto utilizzando un diagramma delle classi e gli opportuni *design pattern*. Inoltre, si descriva utilizzando un diagramma di sequenza l'operazione di scaricamento su un dispositivo di un nuovo libro e la visualizzazione della sua prima pagina.

Soluzione

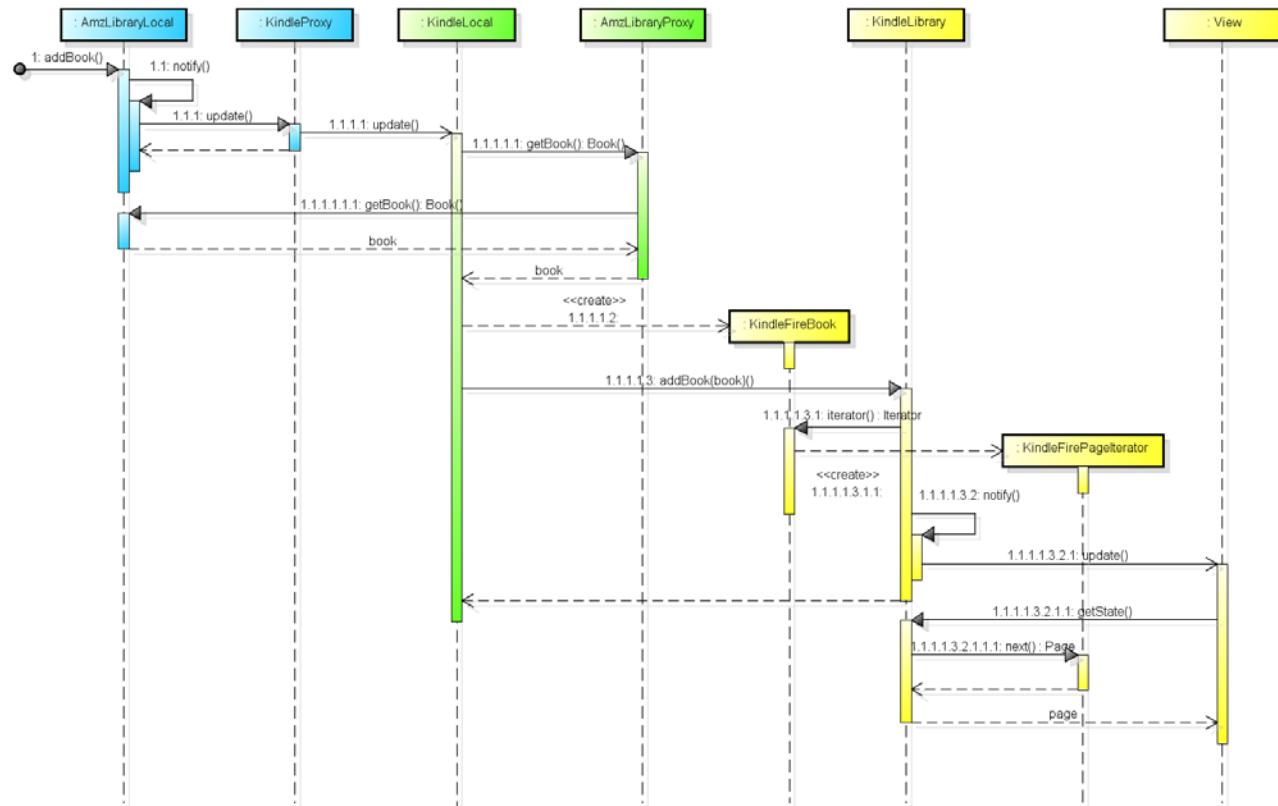
Il diagramma delle classi corrispondente è il seguente, nel quali sono stati utilizzati i pattern:

- Observer
- MVC
- Iterator
- Proxy

È possibile individuare anche un pattern Abstract factory per la creazione delle coppie libro / dispositivo.



Il diagramma di sequenza associato è il seguente.



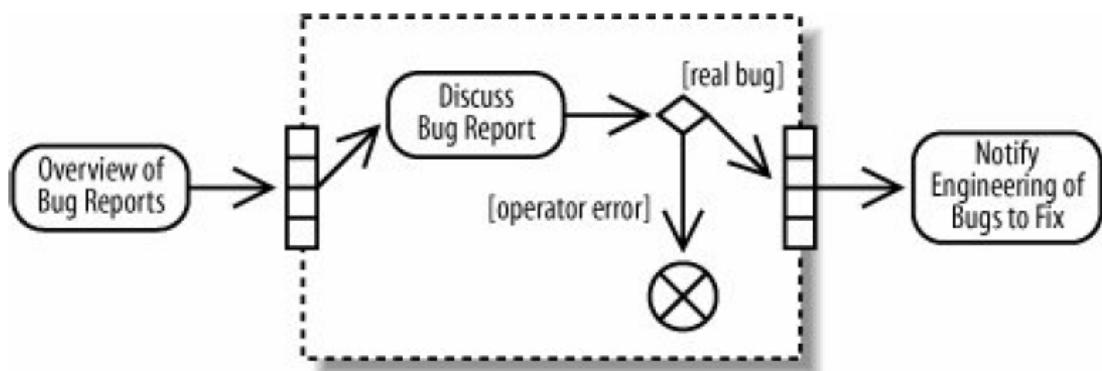
Esercizio 3 (3 punti)

Descrizione

Si fornisca un esempio di diagramma di attività all'interno del quale sia utilizzata una "regione di espansione". Si ricorda che la regione di espansione è la struttura che permette la ripetizione di un insieme di azioni su una collezione di elementi.

Soluzione

Riprendendo l'esempio visto a lezione, una possibile soluzione all'esercizio è la seguente. Dovranno ovviamente essere individuate opportunamente le azioni iniziali e finali.



Ingegneria del Software A.A. 2017/2018

Esame 2018-05-14

Esercizio 1 (6 punti)

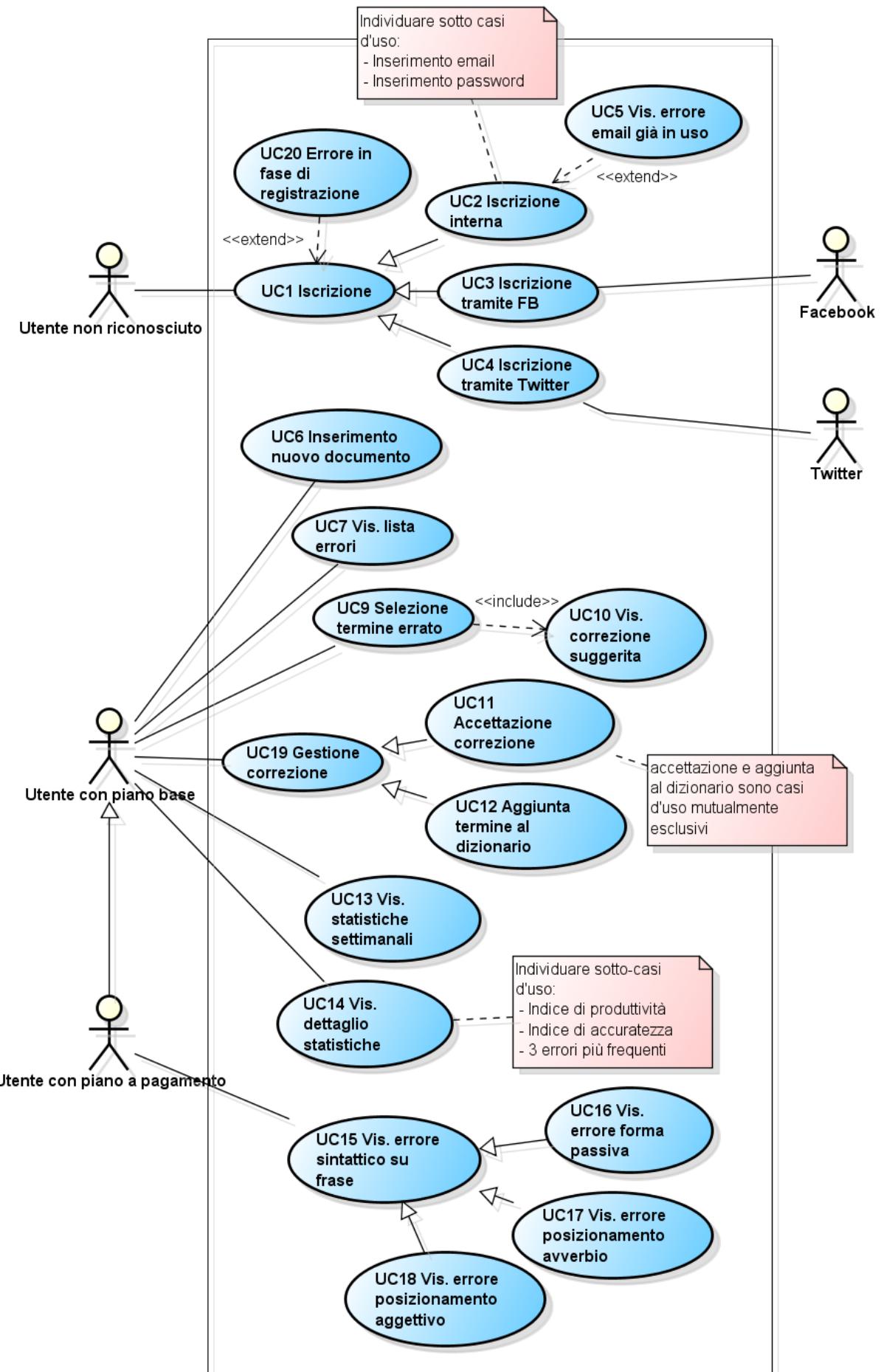
Descrizione

Grammaly è un'applicazione multi-piattaforma, che effettua correzione grammaticale, ortografica, lessicale, e sintattica di testo redatto in inglese. Il servizio si appoggia a un motore in *cloud* per effettuare le opportune analisi sul testo. L'utilizzo dell'applicazione richiede registrazione, tramite una email valida e una *password*, oppure utilizzando un servizio esterno quale Facebook o Twitter. La versione gratuita permette di creare un nuovodocumento, direttamente sul sito, che viene analizzato durante la scrittura. Gli errori rilevati vengono evidenziati con una linea rossa posta sotto la voce errata. Selezionando la segnalazione è possibile visualizzare la correzione suggerita, accettarla, o aggiungere un nuovo termine al proprio dizionario online. Nella sezione relativa al profilo personale è possibile visualizzare le statistiche relative alla propria scrittura. Queste vengono dapprima presentate in una lista ripartita per settimana, visualizzando la settimana di riferimento e la percentuale di accuratezza conseguita in essa. Selezionando un elemento di tale lista, l'utente visualizza le informazioni di dettaglio, quali un indice di produttività, di accuratezza, e i tre errori più frequenti. La versione a pagamento aggiunge l'analisi sintattica del testo, mediante la quale vengono analizzate intere frasi invece che singole voci. Gli errori più comunemente segnalati nelle frasi sono: l'utilizzo di verbi in forma passivo; il posizionamento errato di un avverbio; il posizionamento errato di un aggettivo.

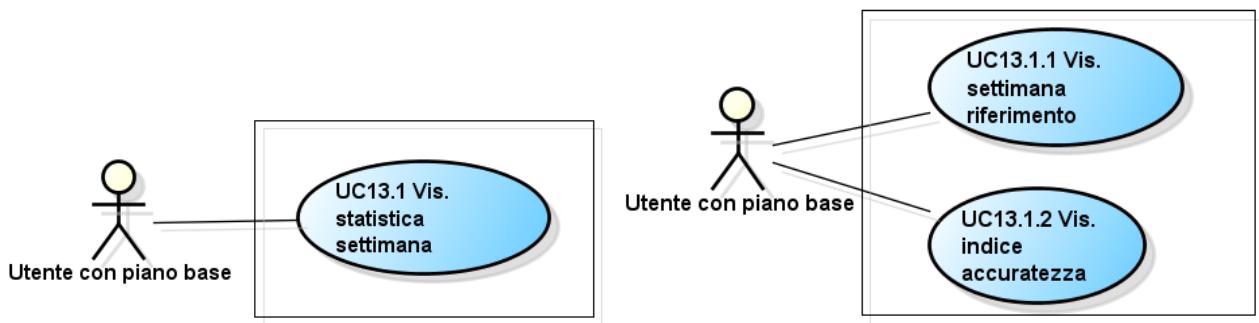
Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari descritti. Non è richiesta la descrizione testuale dei casi d'uso individuati.

Soluzione

La soluzione è la seguente. Ovviamente i sotto-casi d'uso devono essere completamente individuati e modellati in appositi diagrammi.



La visualizzazione delle statistiche settimanali deve essere modellata come una lista.



Esercizio 2 (7 punti)

Descrizione

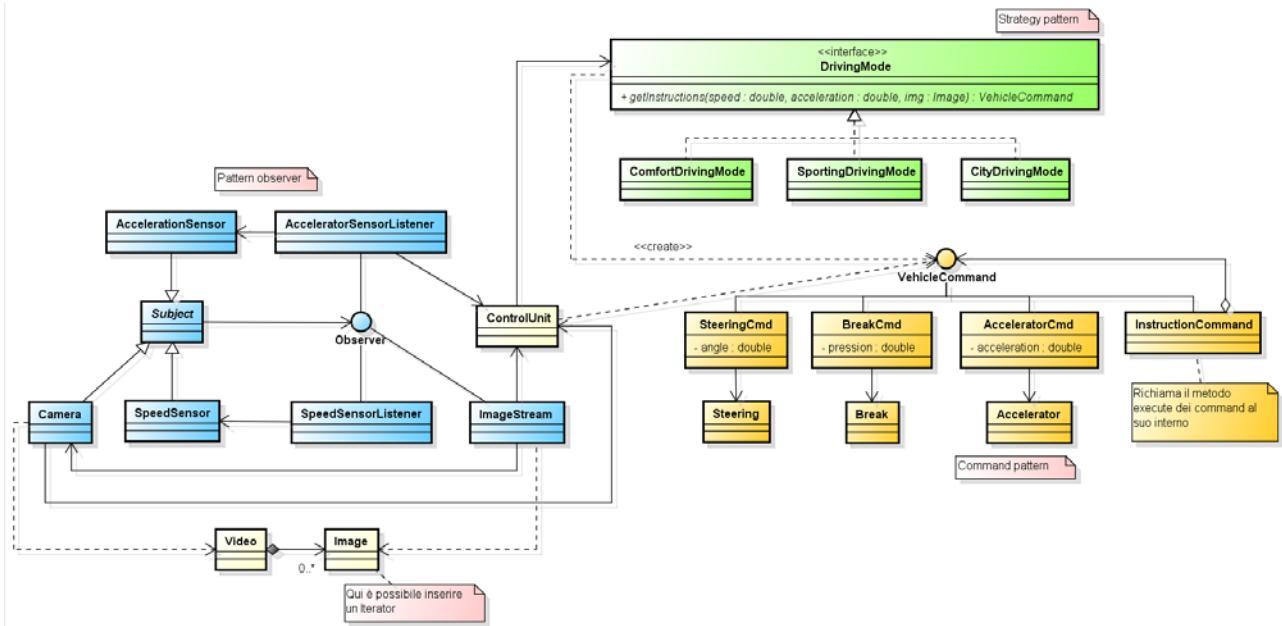
Gli ultimi modelli di auto di alta gamma sono dotati di sistemi di guida autonoma di Livello 3. Con tale dotazione, l'automobile è in grado di gestire la guida in condizioni ambientali ordinarie, gestendo accelerazione, frenata e direzione, in totale autonomia, mentre il guidatore può intervenire in situazioni problematiche, a fronte di richiesta esplicita del sistema o di decisione personale. Per consentire al sistema di guida autonoma di prendere decisioni ben fondate, serve fornirgli un grande mole di dati. Una serie di sensori posti sulla vettura rileva costantemente velocità e accelerazione del veicolo. Una videocamera registra video a 60 fotogrammi/secondo, che vengono poi scomposti in un flusso costante di immagini. Una centralina raccoglie tali informazioni e le elabora, trasformandole in istruzioni per lo sterzo, l'acceleratore e i freni. Per facilitarne la gestione, la centralina genera comandi standard per tipo e formato. Le istruzioni previste sono anche dipendenti dalla modalità di guida scelta: comfort, sportiva,cittadina.

Si modelli tale sistema mediante un diagramma delle classi e i *design pattern* a esso pertinenti. Utilizzando un diagramma di sequenza, si descriva poi il *setting* della velocità in modalità di guida sportiva a fronte della ricezione di una nuova tripletta di informazioni dai sensori.

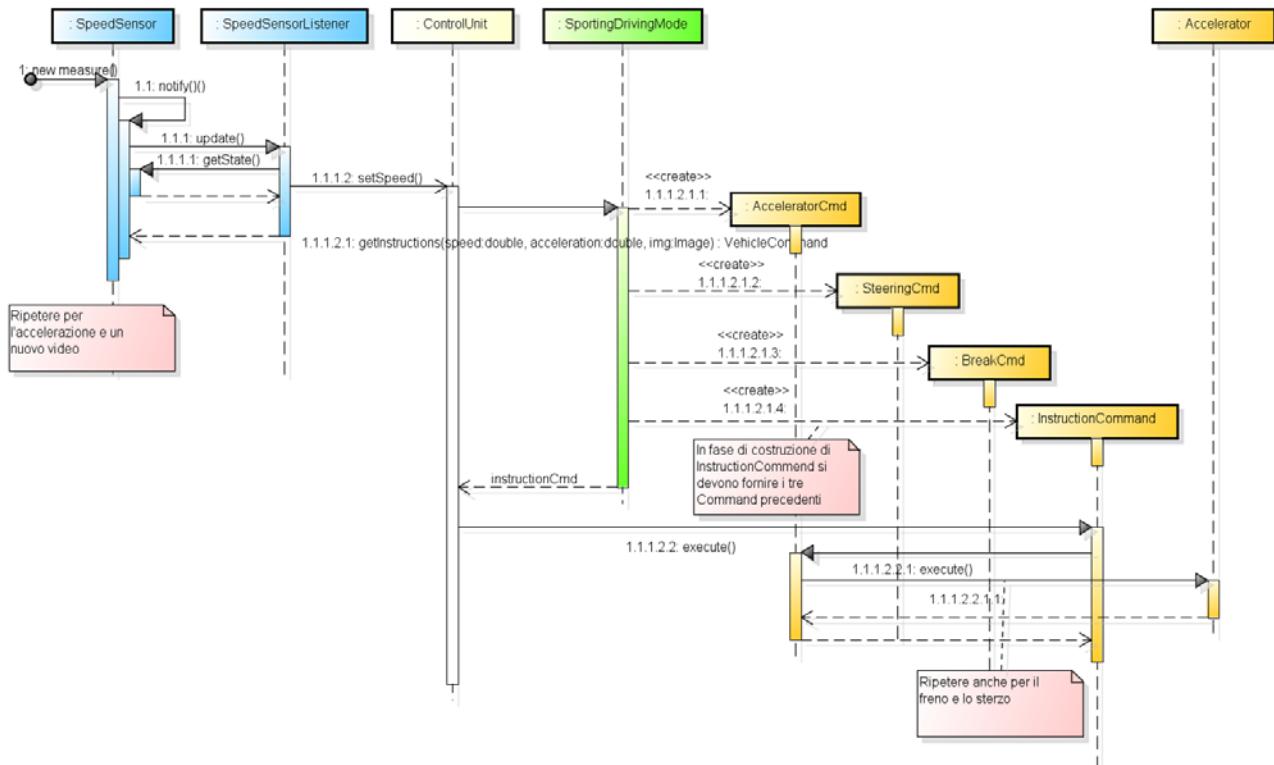
Soluzione

La soluzione coinvolge i seguenti pattern:

- Observer
- (Iterator)
- Strategy
- Command



Il diagramma di sequenza corrispondente è il seguente.



Esercizio 3 (3 punti)

Descrizione

Dato il seguente codice sorgente, derivato da una cattiva progettazione:

```
public class MacBook {
    // Attributi
```

```

public MacBook(String color, String ram, String harddisk,
               String cpu, String graphicCard) { /*...*/ }

public MacBook(String color, String ram, String harddisk,
               String cpu) { /*...*/ }

public MacBook(String color, String ram, String harddisk) { /*...*/ }

public MacBook(String color) { /*...*/ }

// Metodi

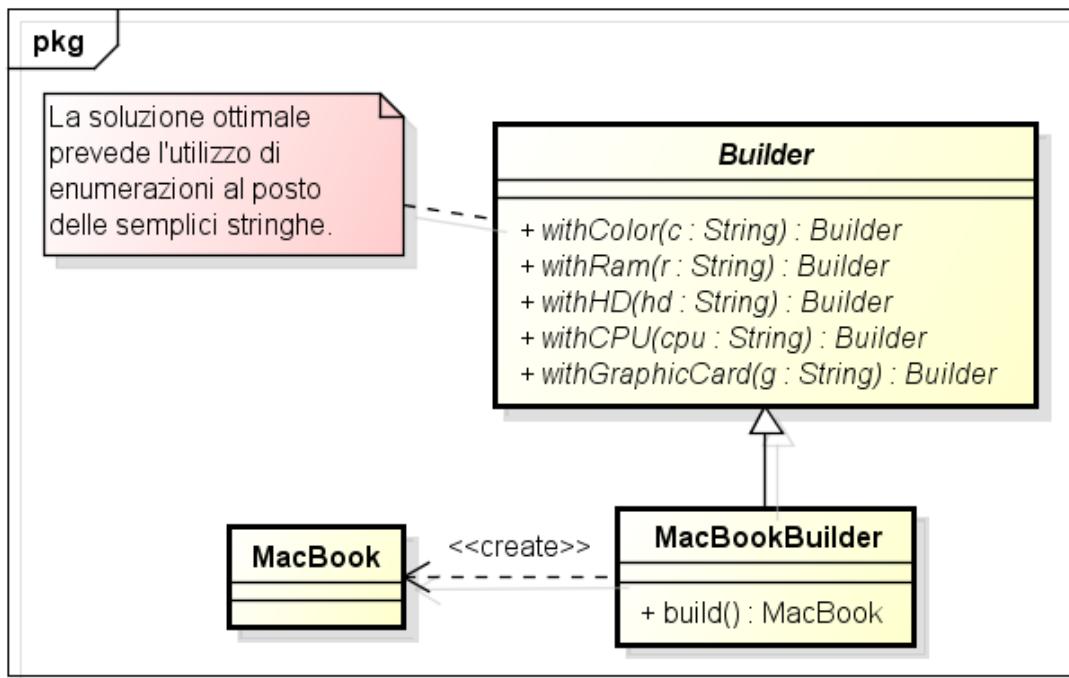
}

```

si fornisca il diagramma delle classi di una sua possibile re-ingegnerizzazione che elimini l'effetto *telescoping* attualmente presente nei costruttori, mantenendo però la caratteristica di selezione delle singole componenti della classe MacBook.

Soluzione

È necessario utilizzare il design pattern Builder.



Ingegneria del Software A.A. 2017/2018

Esame 2018-06-15

Esercizio 1 (6 punti)

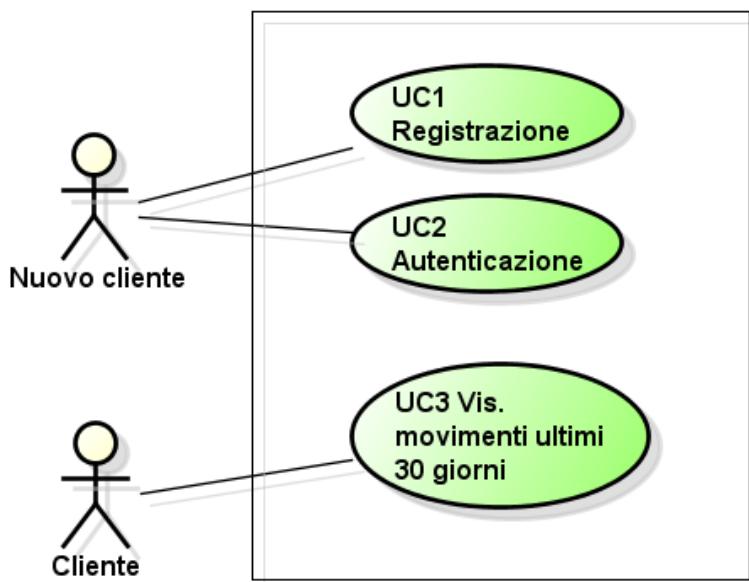
Descrizione

Un sempre maggior numero di banche decidono di fornire ai propri clienti la possibilità di aprire una nuova posizione (conto corrente) senza doversi recare in filiale. Tramite l'utilizzo di foto e videochiamate, è possibile eseguire un processo di riconoscimento sicuro. In particolare, un nuovo cliente, nella fase di registrazione, deve inserire d'apprima un email valida e il suo numero di cellulare. Successivamente, il sistema invia due codici distinti, uno all'email e uno al numero di cellulare, che l'utente deve inserire per confermare la propria identità. Successivamente, il sistema richiede di effettuare una foto al proprio tesserino sanitario e alla propria carta di identità. Da questi documenti, vengono desunte le generalità del nuovo cliente, tra le quali nome, cognome, codice fiscale e numero di documento di identità. Queste vengono visualizzate in una pagine riassuntiva. Una volta inserite le suddette informazioni, viene iniziata una video chiamata con un operatore, durante la quale l'operatore stesso effettua una verifica più approfondita dei dati inseriti in precedenza. Una volta che la video chiamata è terminata con successo, è possibile autenticarsi all'applicazione web delle banca, utilizzando le credenziali appena fornite dall'operatore. Essendo la posizione aperta una posizione bancaria valida a tutti gli effetti, il sito della banca permette di vedere i propri movimenti degli ultimi 30 giorni. Per ogni movimento di uscita nella lista, vengono visualizzati la data della valuta, il beneficiario e l'importo.

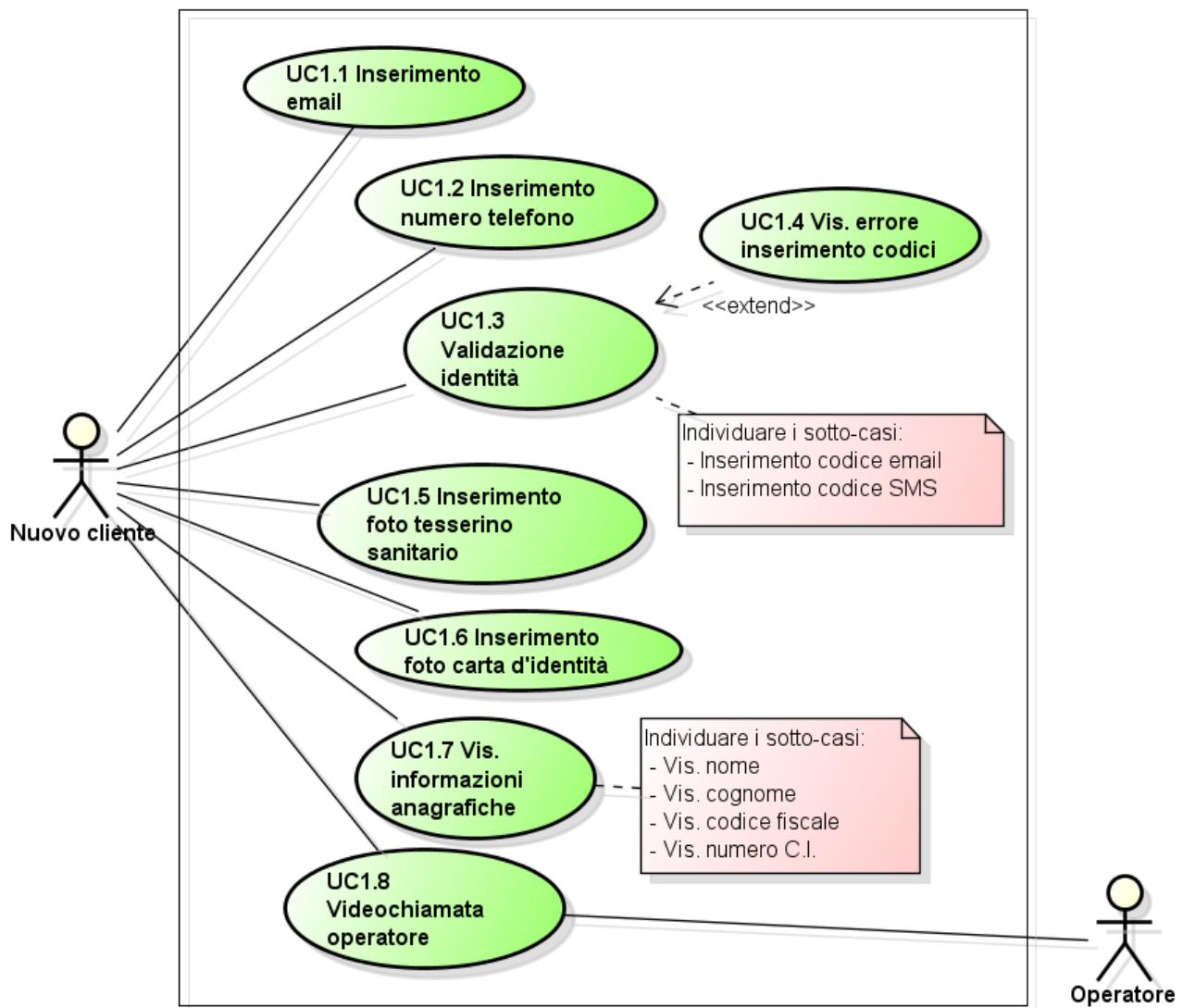
Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari descritti. Non è richiesta la descrizione testuale dei casi d'uso individuati.

Soluzione

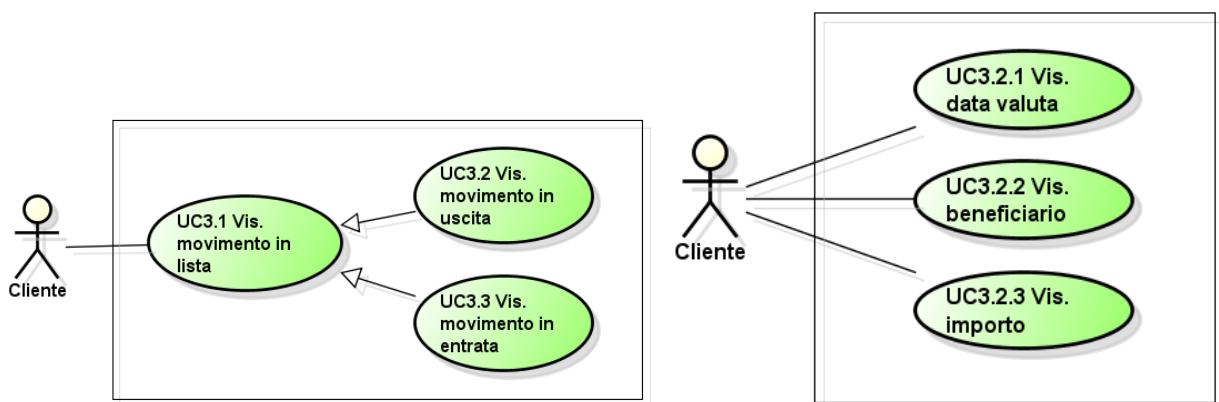
La soluzione parte con la differenziazione dei casi d'uso dei nuovi clienti e dei clienti già riconosciuti.



La registrazione può essere dettagliata come segue.



La visualizzazione dei movimenti in uscita si può modellare come segue.



Esercizio 2 (7 punti)

Descrizione

Spotify è una nota applicazione per l'ascolto di musica in streaming ed *offline*. I server di Spotify forniscono lo streaming musicale utilizzando una struttura remota a pacchetti (*chunk*). Una canzone è composta da più *chunk*. Ogniqualvolta un nuovo *chunk* è disponibile durante uno streaming audio, l'applicazione client connessa ne viene opportunamente informata. Un brano può essere riprodotto con qualità standard, pari a 192Kbps, oppure con qualità elevata, pari a 320Kbps. I *chunk* vengono man mano decodificati in da un algoritmo opportunamente scelto in istruzioni per la scheda audio. Spotify, però, fornisce anche la modalità di ascolto *offline*, durante la quale i brani vengono letti da filesystem. Seppur la libreria utilizzata per la lettura del file, utilizzi un semplice array di byte per rappresentare il file audio, l'applicazione adatta tale informazione alla modalità di lettura a *chunk*.

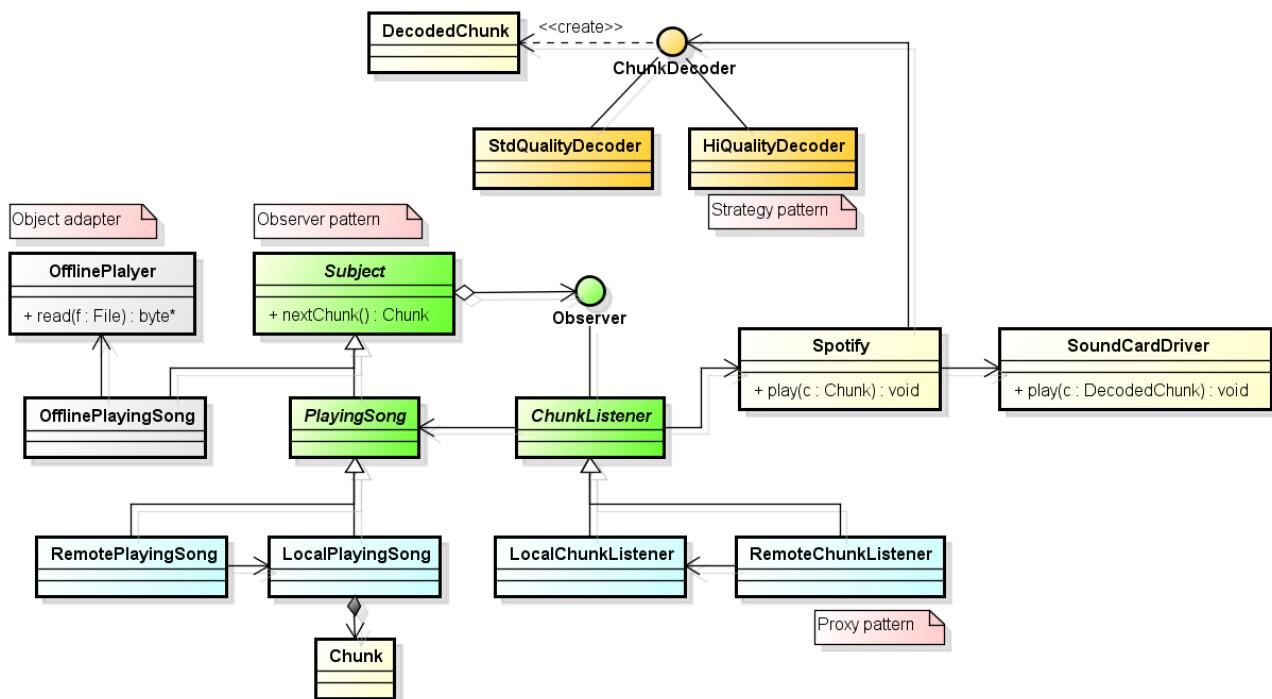
Si modelli tale sistema mediante un diagramma delle classi e i *design pattern* a esso pertinenti. Utilizzando un diagramma di sequenza, si descriva poi l'arrivo di un nuovo *chunk* remoto e il suo processamento da parte del sistema in istruzioni per la scheda audio.

Soluzione

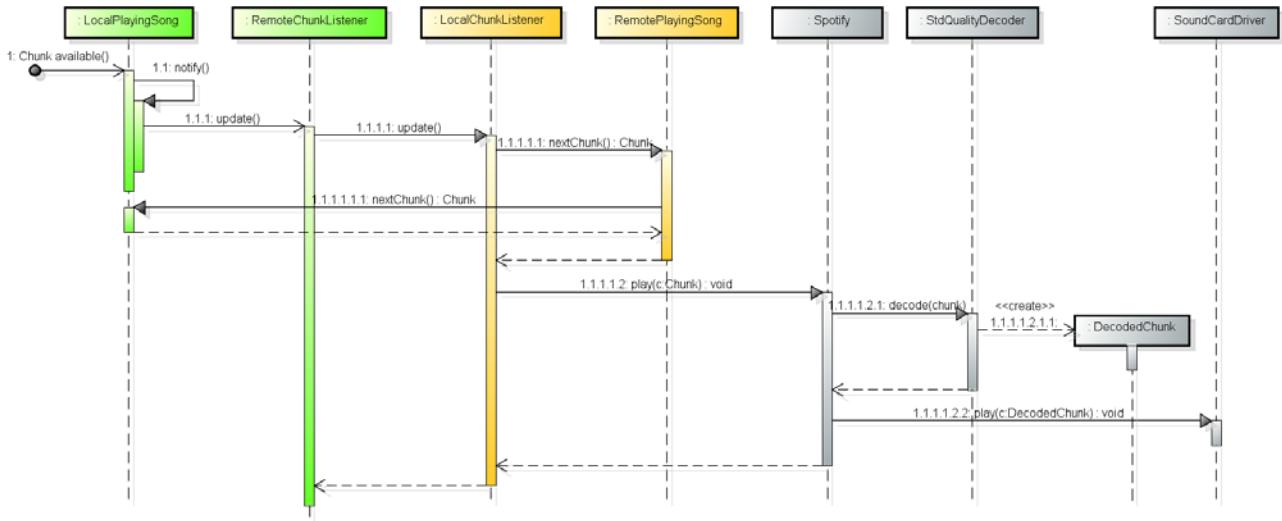
I design pattern da utilizzare sono i seguenti:

- Observer pattern
- Proxy pattern
- Object adapter pattern
- Strategy pattern

Il diagramma delle classi corrispondente è il seguente.



Il diagramma di sequenza è invece il seguente.



Esercizio 3 (3 punti)

Descrizione

Sia dato il seguente frammento di codice sorgente in Java.

```

public class Animal {
    private String animalType;

    public static void letsTalk(Animal[] animals) {
        for (Animal a: animals) {
            switch (a.getAnimalType) {
                case "Dog":
                    ((Dog) a).bark();
                    break;
                case "Cat":
                    ((Cat) a).mew();
                    break;
            }
        }
    }
    // Other methods...
}

public class Dog {
    // Code that properly initialize super animalType
}

```

```

public void bark() { System.out.println("Bark bark"); }

}

public class Cat {

    // Code that properly initialize super animalType

    public void miaow() { System.out.println("Mew mew"); }

}

```

Si modifichi il suddetto codice al fine di fare aderire il metodo letsTalk al principio Open-Closed.

Soluzione

È necessario uniformare i metodi nelle classi concrete in una unica interfaccia talk, dichiarata in Animal. In questo modo, è possibile estendere il sistema aggiungendo nuovi animali, senza tuttavia modificare il codice esistente.

```

public abstract class Animal {

    public abstract void talk();

    public static void letsTalk(Animal[] animals) {

        for (Animal a: animals) {

            a.talk();

        }
    }
}

public class Dog extends Animal {

    @Override

    public void talk() { System.out.println("Bark bark"); }

}

public class Cat extends Animal {

    @Override

    public void talk() { System.out.println("Mew mew"); }

}

```

Ingegneria del Software A.A. 2017/2018

Esame 2018-07-20

Esercizio 1 (6 punti)

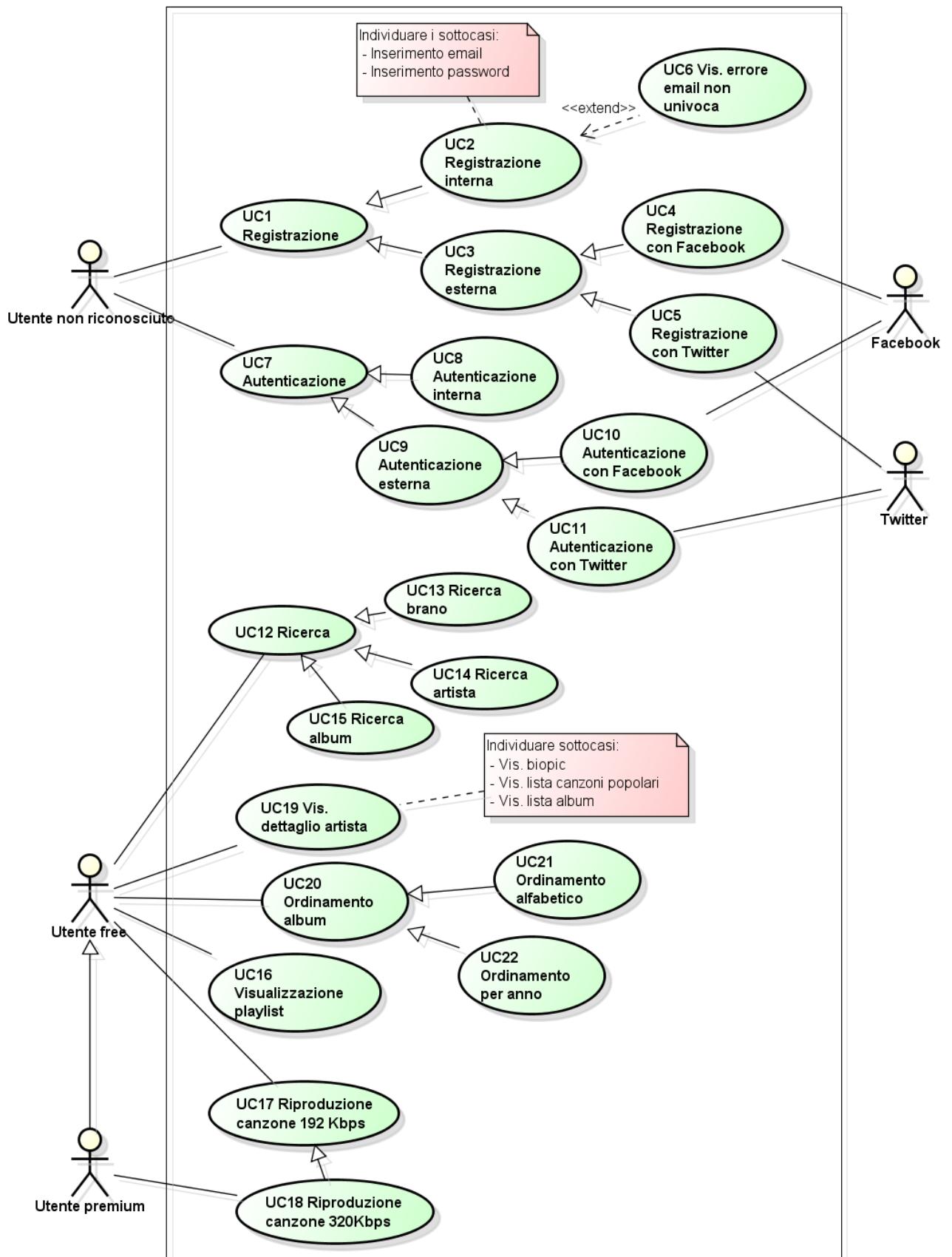
Descrizione

Spotify è una famosa applicazione per l'ascolto di musica, sia in *streaming* che *off-line*. L'iscrizione al sito avviene utilizzando tre modalità differenti: inserendo un indirizzo di posta elettronica univoco e una *password*, utilizzando il proprio *account* Facebook o Twitter. Dopo aver effettuato l'accesso, l'utente può ricercare un brano musicale, un artista o un album per nome, oppure scegliere tra le *playlist* fornite da Spotify. Ogni *playlist* visualizza una lista di canzoni. Per ognuna di esse, la visualizzazione presenta il nome, l'artista, e la durata. Tramite un opportuno tasto è inoltre possibile avviare la riproduzione di una singola canzone. Il *bitrate* base della riproduzione è fissato a 192 Kbps per gli utenti *free*; gli utenti premium possono aumentarlo a 320 Kbps. È disponibile inoltre un dettaglio di ogni artista, che ne visualizza un *biopic*, una lista delle sue canzoni più popolari e la lista dei suoi album. Questi ultimi possono essere ordinati alfabeticamente o per anno di uscita.

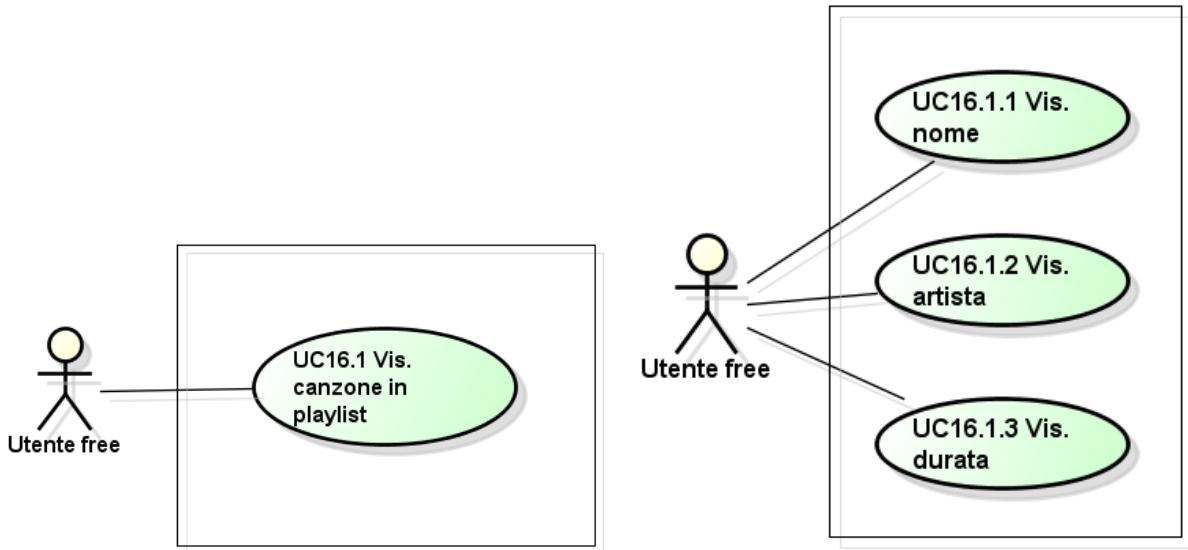
Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari sopra descritti. Non ne è richiesta la descrizione testuale.

Soluzione

La soluzione è la seguente.



Nel dettaglio, la visualizzazione di una *playlist* viene invece modellata dai seguenti casi d'uso.



Esercizio 2 (7 punti)

Descrizione

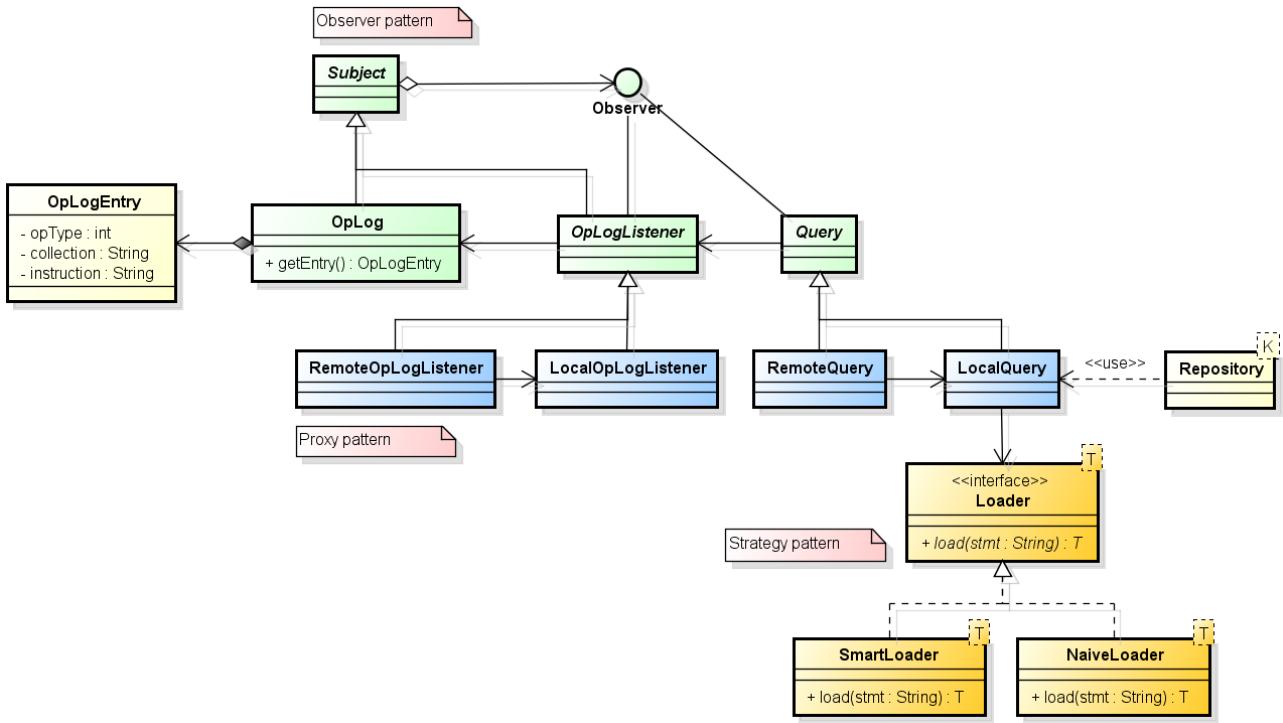
I nuovi driver di connessione a basi dati NoSQL e NewSQL sposano il paradigma *reactive* e il modello a eventi. Il *driver rx-mongo* per il DB NoSQL MongoDB è un esempio di questa tendenza. MongoDB è un DB documentale, che tratta i documenti come *file JSON*, e li indica con un campo riservato (*field*) chiamato “*_id*”. Una componente di tale *driver* resta in ascolto delle modifiche dell’*Oplog* di Mongo, il *file* dove vengono riportate tutte le operazioni di scrittura che avvengono sul nodo *master* del DB. A ogni modifica di *collection* (insieme di documenti) effettuata da un’operazione di scrittura, tale componente notifica tutte le *query* che si sono nel frattempo registrate per ricevere aggiornamenti. Per ogni scrittura, essa riporta i *field* che sono stati oggetto di modifica. In questo modo, la *query* registrata può scegliere se rileggere i dati dal DB o meno. La rilettura avviene utilizzando una apposita struttura, della quale esistono differenti implementazioni. La versione base ripete la *query* sul DB. La versione più sofisticata usa algoritmi avanzati per calcolare come i dati possano essere cambiati. La *query* e la componente di ascolto delle modifiche sono naturalmente dislocate in punti differenti della rete. Una volta che la *query* ha recuperato i dati aggiornati, essi vengono poi forniti ai *repository* che le hanno dichiarate.

Si modelli tale sistema mediante un diagramma delle classi e i *design pattern* a esso pertinenti. Utilizzando un diagramma di sequenza, si descriva poi l’*update* dell’indirizzo di residenza relativo a un documento di una *collection*, e l’aggiornamento di una *query* che recupera tutti i documenti della *collection*.

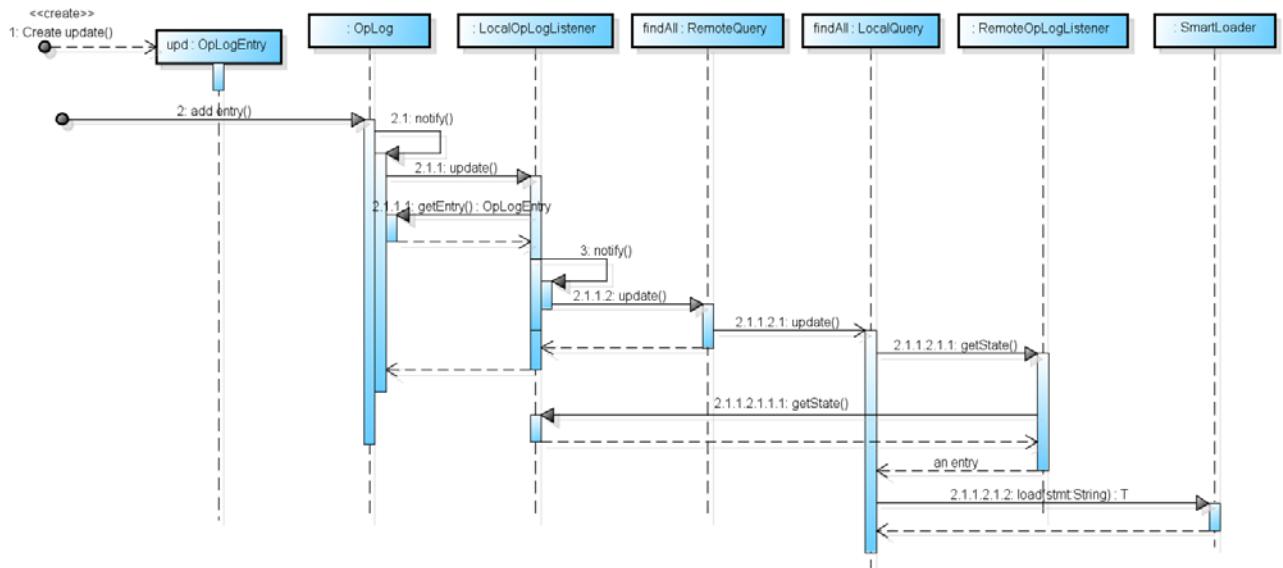
Soluzione

La soluzione prevede l’utilizzo dei seguenti design pattern:

- Observer pattern
- Proxy pattern
- Strategy pattern



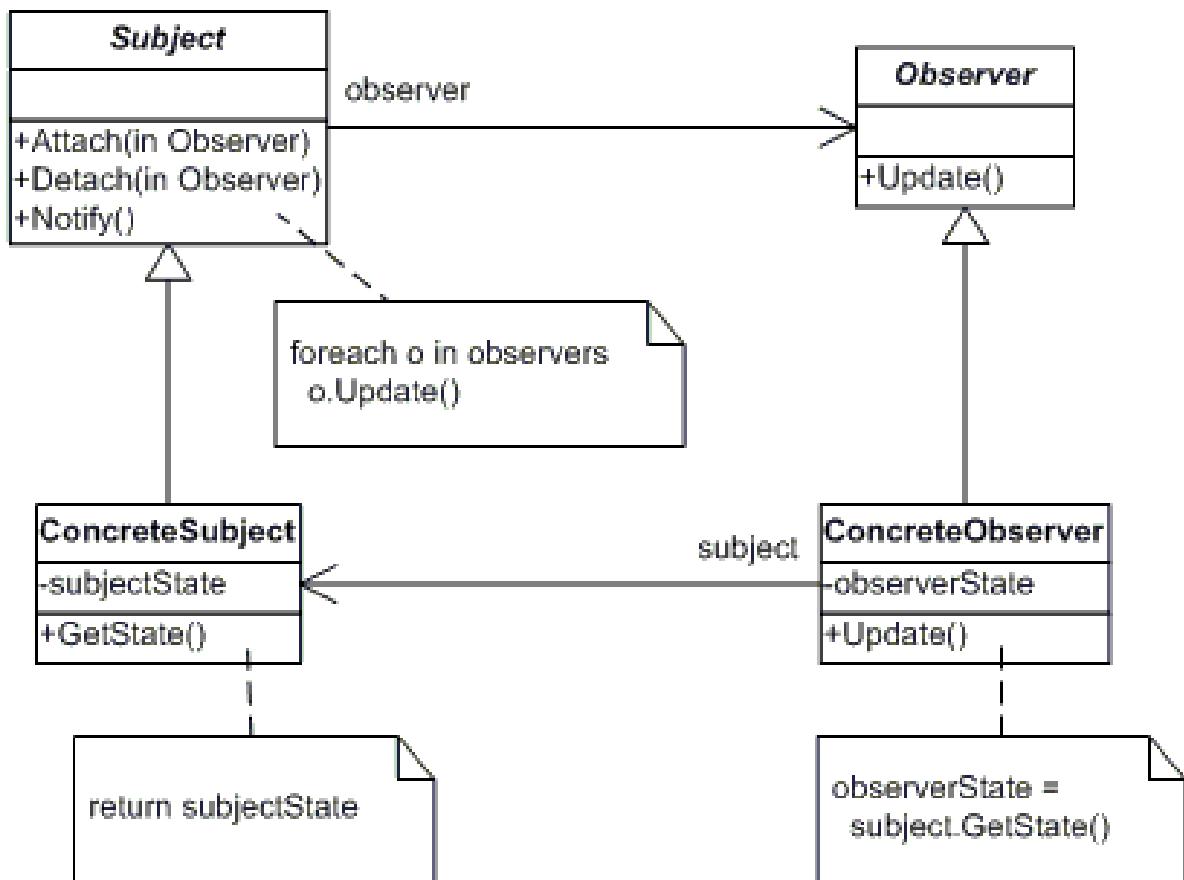
Il diagramma di sequenza richiesto è invece il seguente.



Esercizio 3 (3 punti)

Descrizione

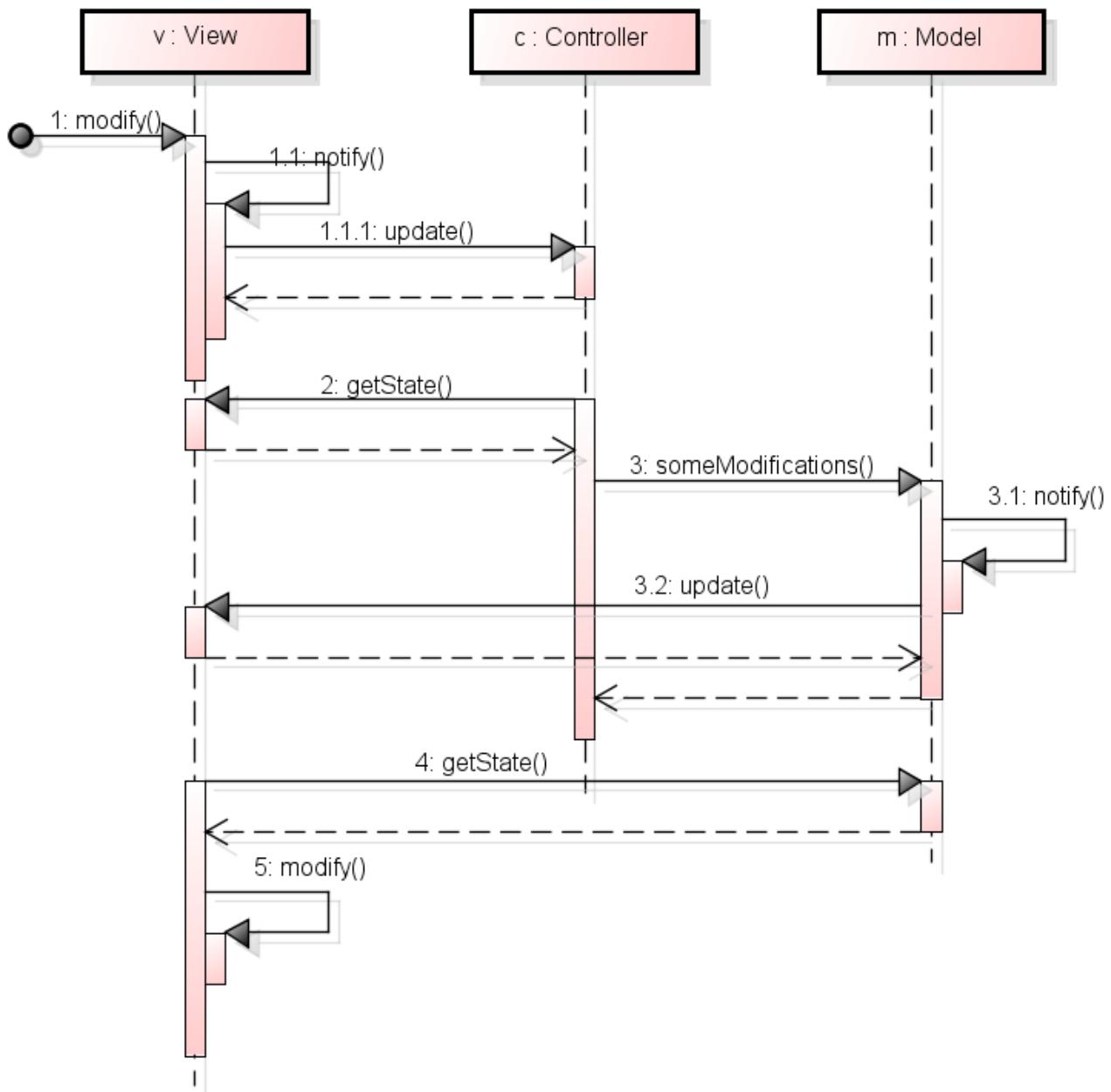
Nella sua versione base, nota come *push model*, il pattern Model View Controller, MVC, viene implementato utilizzando una doppia istanza del *pattern Observer*, rappresentato nel diagramma delle classi sotto riportato.



Utilizzando un diagramma di sequenza, si modelli lo scambio di messaggi fra le componenti del *pattern MVC*, corrispondente al seguente flusso di operazioni: l'utente effettua un'operazione sulla vista, che modifica il modello e, di conseguenza, le informazioni visualizzate nella vista stessa.

Soluzione

Utilizzando un doppio pattern Observer, le componenti Model, View e Controller interagiscono nel seguente modo.



Ingegneria del Software A.A. 2017/2018

Esame 2018-08-24

Esercizio 1 (6 punti)

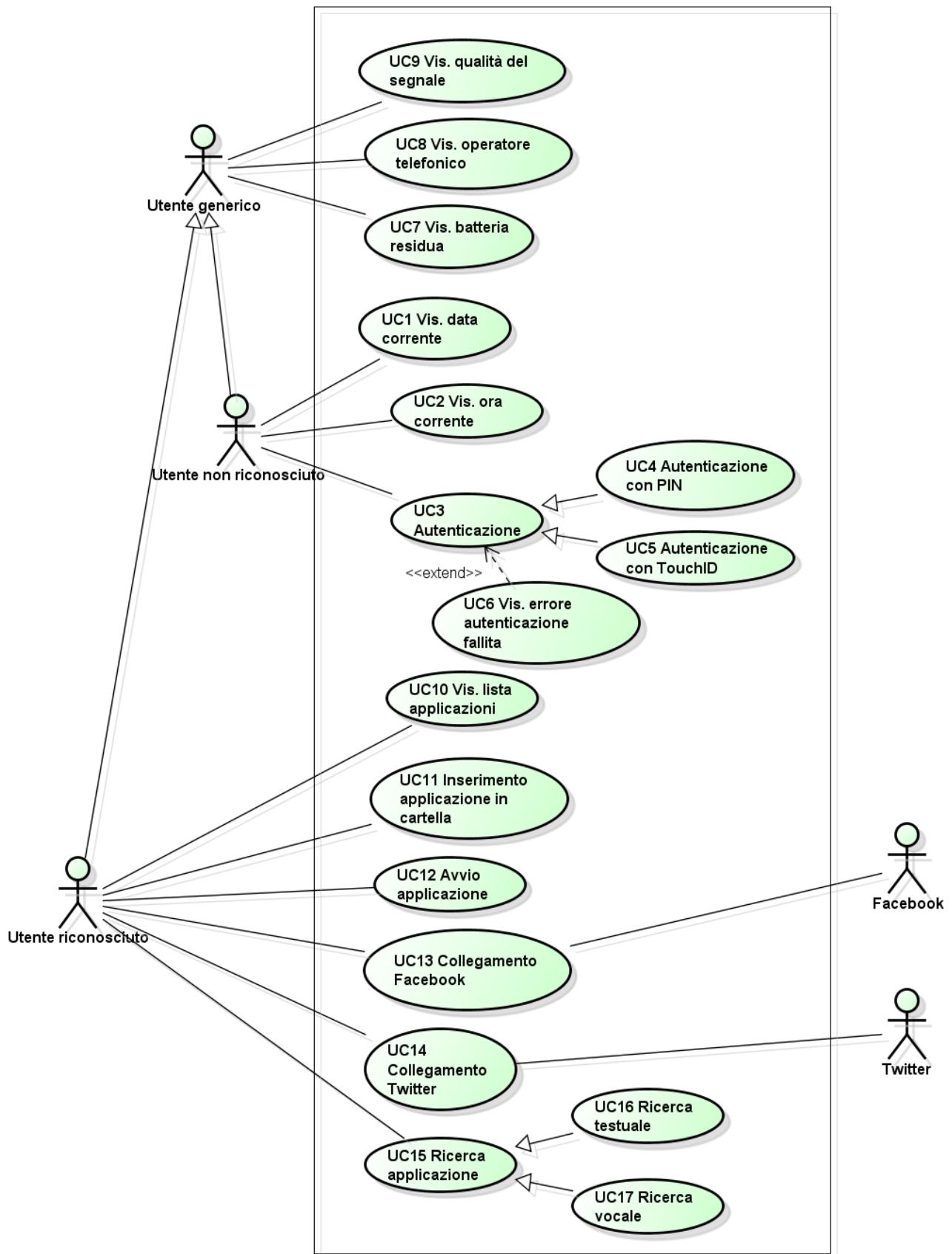
Descrizione

Il sistema operativo iOS è il cuore dell'iPhone, lo *smartphone* di punta di Apple. La pagina iniziale del sistema riporta l'ora e la data del giorno corrente. Da questa pagina, l'utente può decidere di autenticarsi utilizzando un codice di 6 cifre oppure identificarsi tramite impronte digitali se tale funzione è stata precedentemente abilitata. Un *pop-up* di errore notifica l'eventuale inserimento di informazioni errate. Dopo tre riconoscimenti errati delle impronte digitali, la funzionalità viene disabilitata fino al corretto inserimento del codice di 6 cifre. Alcune informazioni sono presenti sia prima che dopo il riconoscimento dell'utente; ad esempio, la qualità di segnale della rete cellulare, l'operatore telefonico che fornisce il segnale, e la quantità di batteria residua. Una volta riconosciuto, l'utente ha accesso alla sua pagina *home*, che riporta la lista delle applicazioni disponibili. Di ognuna di esse è visualizzato il nome. Le applicazioni possono essere raggruppate in cartelle dall'utente, modificando in questo modo la loro visualizzazione nella pagina *home*: inizialmente, sarà visualizzata una cartella con associato un nome, che una volta selezionata, visualizzerà la lista dettagliata di applicazione al suo interno. La selezione di una singola applicazione, in entrambi i casi, ne inizia l'esecuzione. È possibile collegare il proprio *account* Facebook e Twitter all'interno dell'area Impostazioni, in modo da facilitare la condivisione di contenuti sulle reti sociali. Infine, è possibile ricercare un'applicazione digitando il suo nome, o pronunciandoli, utilizzando i comandi vocali.

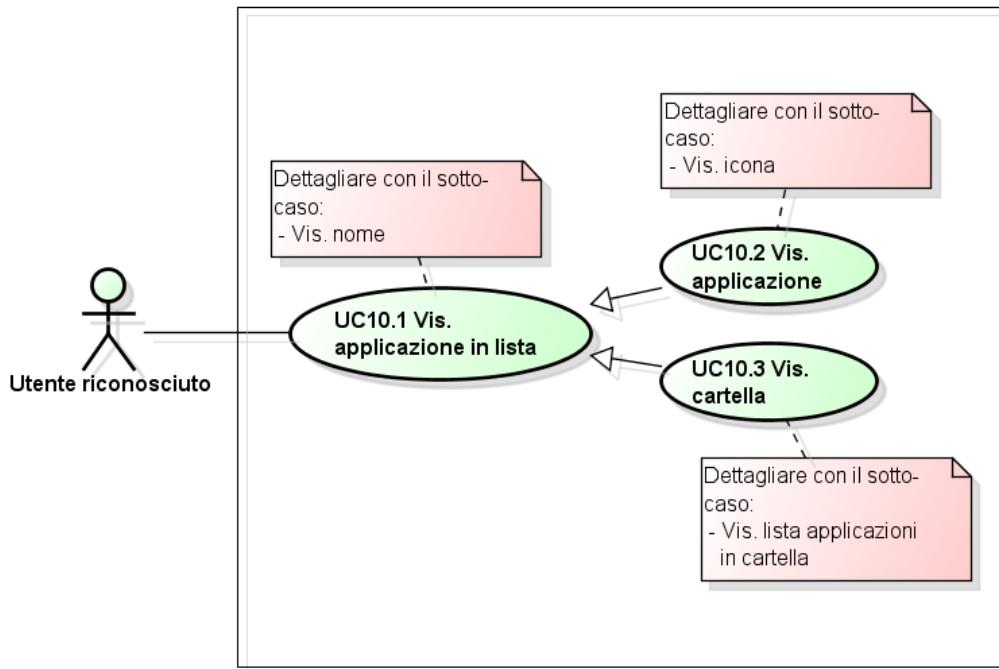
Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari sopra descritti. Non ne è richiesta la descrizione testuale.

Soluzione

Il diagramma dei casi d'uso che modella una possibile soluzione al problema è il seguente.



Nel dettaglio, UC10 individua i seguenti sotto-casi d'uso.



Esercizio 2 (7 punti)

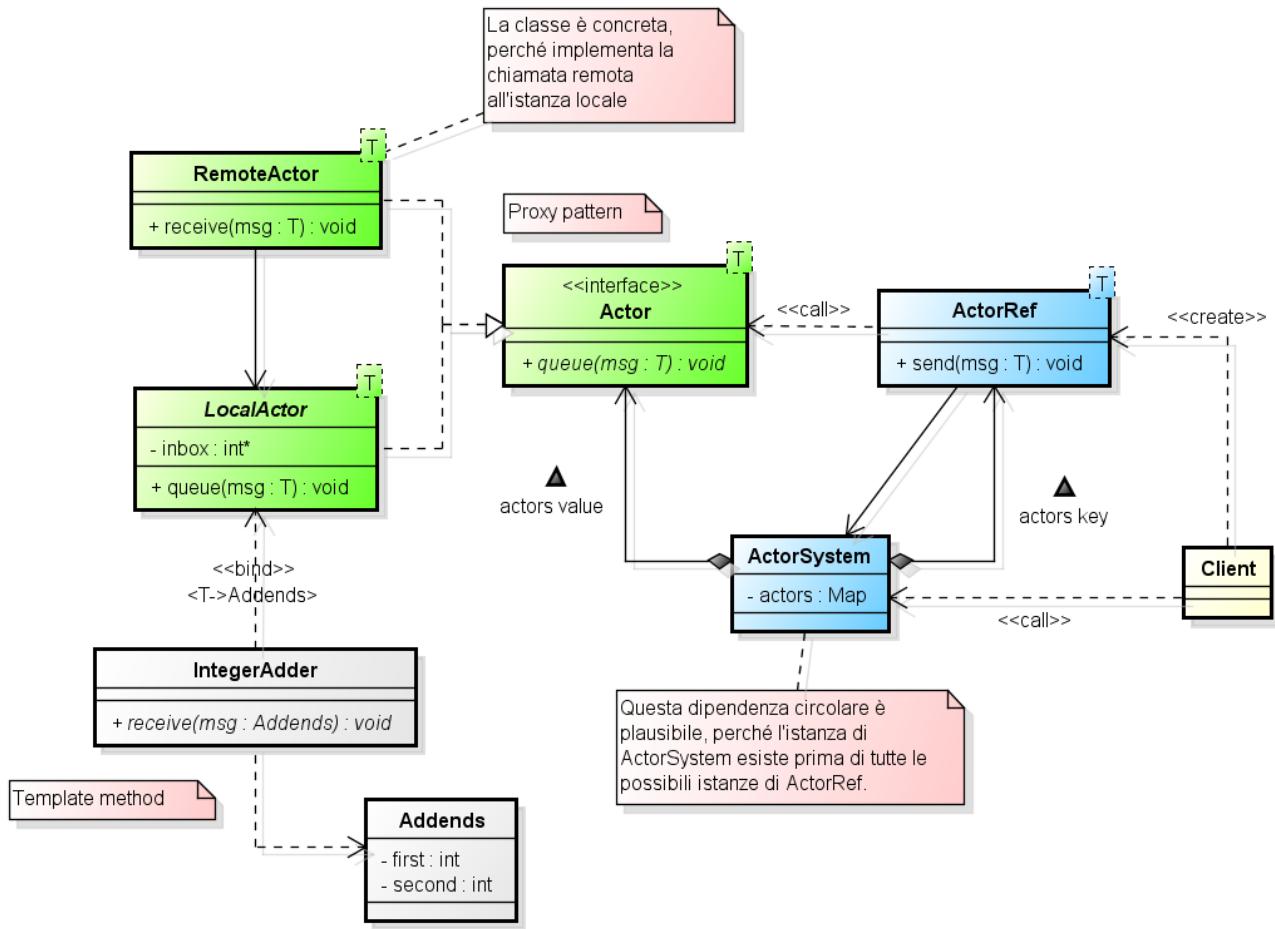
Descrizione

Uno dei possibili modelli di esecuzione distribuita di un programma è il cosiddetto “modello ad attori”. Ogni attore, `Actor`, rappresenta una componente a sé stante, che può ricevere messaggi da altri attori e può crearne a sua volta. L’attore legge i messaggi da una `inbox` e reagisce a questi eseguendo attività ad essi corrispondenti. Tali funzionalità vengono realizzate all’interno di un metodo `receive`, che reagisce a messaggi di tipo generico `T`. Per inviare un messaggio a un attore non si utilizza direttamente un riferimento a un’istanza concreta della classe `Actor`, ma piuttosto alla classe `ActorRef`, che modera l’accesso all’istanza concreta. Il sistema di attori è orchestrato da un *actor system*, che mantiene in una mappa associativa una copia di tutti gli attori e dei loro corrispondenti riferimenti, in modo tale che ogni `ActorRef` sia associato a un `Actor`, locale o remoto. Per permettere al tutto di funzionare, ogni `ActorRef` possiede un riferimento all’*actor system*.

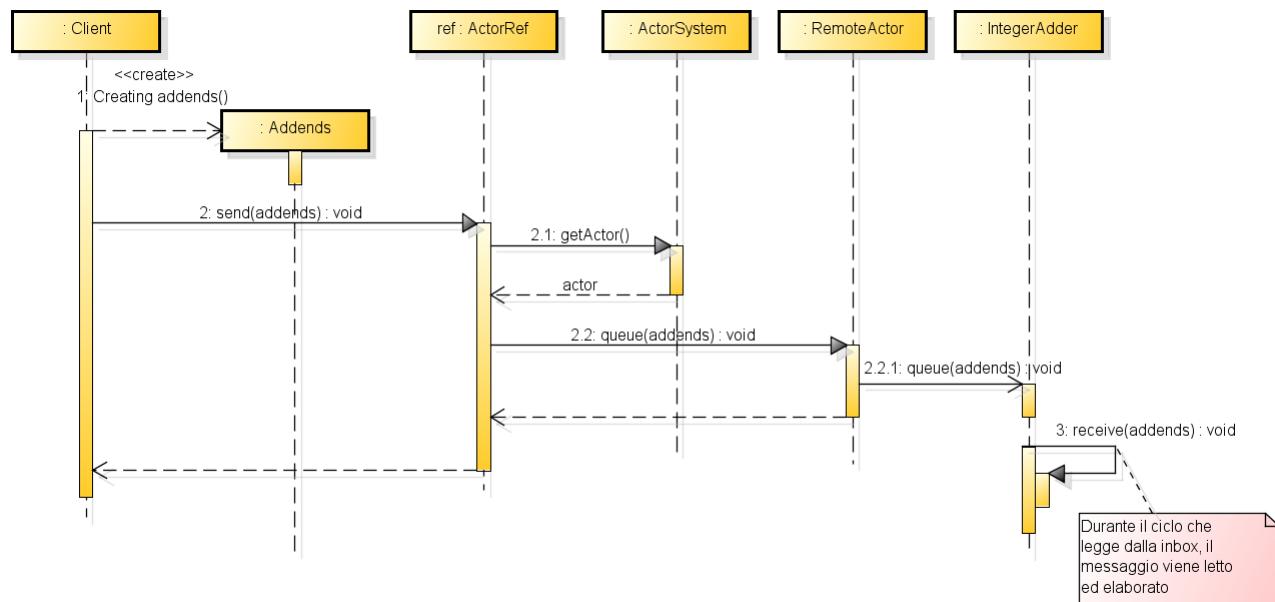
Si modelli tale sistema mediante un diagramma delle classi ed i *design pattern* a esso pertinenti. Utilizzando un diagramma di sequenza, si descriva l’invio di un messaggio contenente due interi ad un attore, che ne effettui la somma.

Soluzione

La soluzione più semplice è la seguente, dove si individua un *Proxy pattern* ed eventualmente un *Template Method* con le implementazione della classe `LocalActor`.



Il diagramma di sequenza richiesto è il seguente. Si noti che la comunicazione all'interno del pattern proxy è chiaramente asincrona.



Esercizio 3 (3 punti)

Descrizione

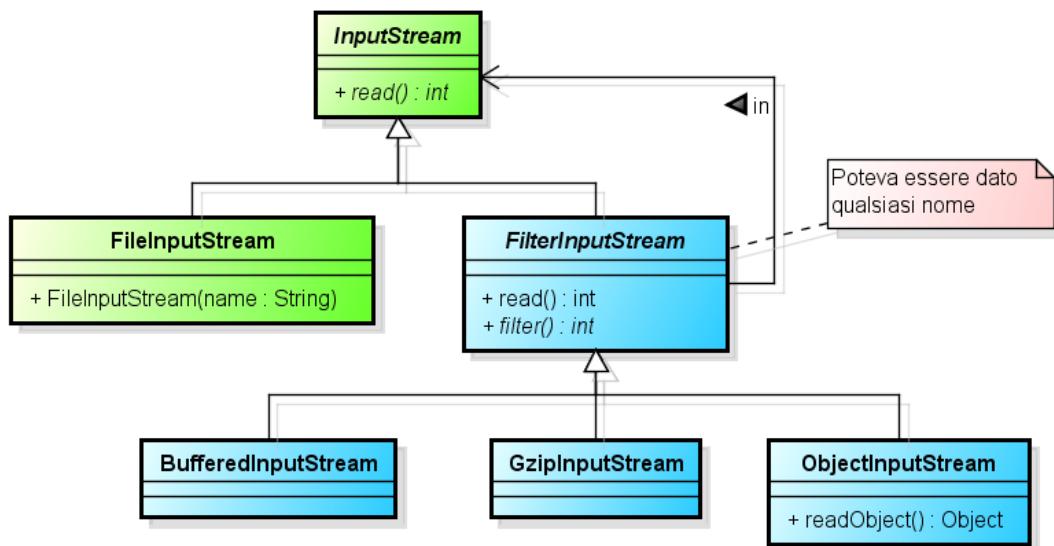
Il *framework* degli I/O stream in Java ha le classi astratte `InputStream` e `OutputStream` come classi base. La componente di *input*, `InputStream`, espone un unico metodo astratto, `read(): int`, che ritorna il prossimo *byte* di dati da leggere. Usando le classi messe a disposizione da questo *framework* è possibile leggere un oggetto precedentemente salvato in un *file* compresso, utilizzando il seguente frammento di codice.

```
FileInputStream fis = new FileInputStream("/objects.gz");
BufferedInputStream bis = new BufferedInputStream(fis);
GzipInputStream gis = new GzipInputStream(bis);
ObjectInputStream ois = new ObjectInputStream(gis);
SomeObject someObject = (SomeObject) ois.readObject();
```

Il *framework* degli I/O stream utilizza infatti un noto *design pattern* della GoF. Si fornisca il diagramma delle classi che contestualizza il *pattern* nella gerarchia di classi sopra riportata.

Soluzione

Il *design pattern* da utilizzare è il Decorator pattern. La classe decorator è identificata in `FilterInputStream`, ma poteva essere fornito qualsiasi nome.



Ingegneria del Software A.A. 2016/2017

Esame 2017-09-11

Esercizio 1 (6 punti)

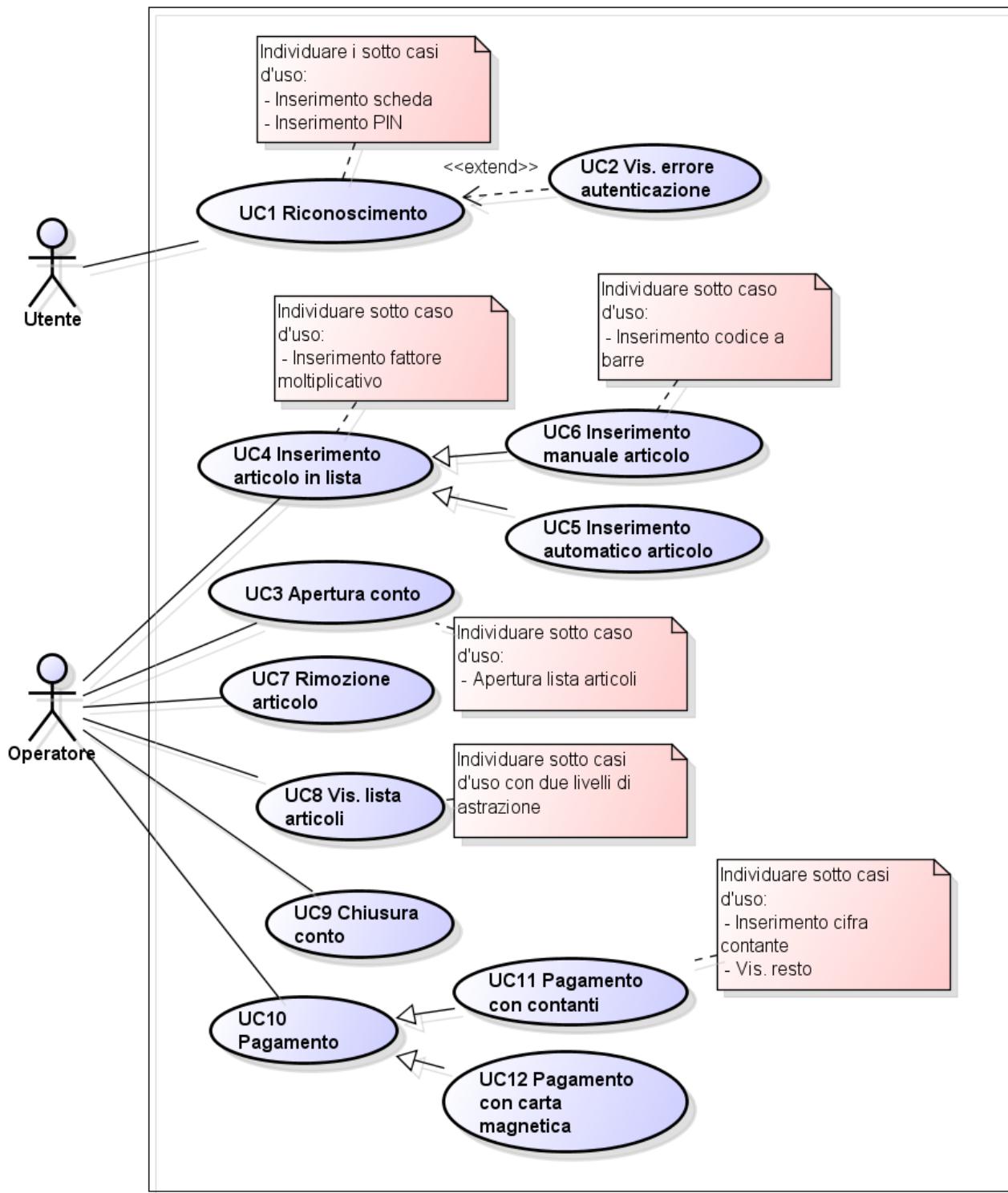
Descrizione

I programmi software installati nei registratori di cassa elettronici diventano sempre più sofisticati con l'evolvere della tecnologia. Per poter accedere alle funzionalità offerte da un regista di cassa moderno, l'operatore deve farsi riconoscere utilizzando una scheda magnetica, che riporta – tra l'altro – la username dell'operatore. Questi deve poi inserire a tastiera un PIN di 5 cifre che vale come password. L'errore di l'inserimento password causa la visualizzazione di un opportuno messaggio. Una volta riconosciuto, l'operatore può accedere a numerose funzionalità. Innanzitutto, può iniziare un nuovo conto per un cliente. Questa operazione consente di iniziare a creare la lista di articoli comprati, passando il codice a barre del prodotto sull'apposito lettore. Questa operazione contatta il sistema centralizzato del negozio che, accedendo a un DB, ritorna le informazioni di dettaglio del prodotto. È possibile rimuovere un articolo dalla lista del conto o inserirne uno digitando manualmente il codice a barre. Per l'acquisto di più unità dello stesso prodotto, è possibile specificare un fattore moltiplicativo intero da applicare al prezzo unitario. A fine conto, è possibile visualizzare la lista degli articoli inseriti. Tale lista, per ogni articolo riporta le seguenti informazioni: numero, breve descrizione e prezzo complessivo della riga. Il pagamento di un conto finale può avere tramite contanti, carta di credito o debito. Nel primo caso, l'operatore inserisce l'importo del contante ricevuto dal cliente e riceve in risposta l'eventuale resto.

Si utilizzino diagrammi dei casi d'uso per modellare gli scenari sopra descritti. Non ne è richiesta la descrizione testuale.

Soluzione

Individuare e modellare esplicitamente i sotto casi d'uso ove richiesto.



Esercizio 2 (7 punti)

Descrizione

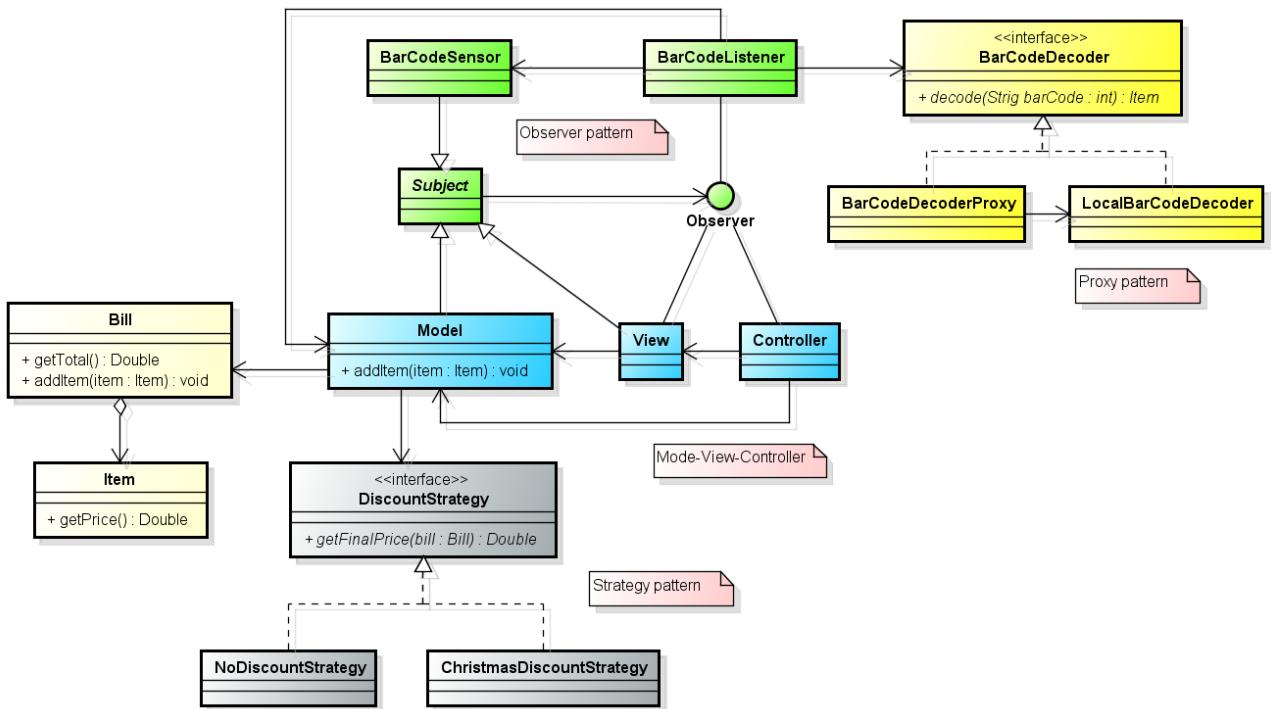
Il software installato in un registratore di cassa come quello descritto nell'esercizio 1, è abbastanza complesso da dover essere progettato utilizzando best practice architetturali. Oltre al software che gestisce l'interfaccia utente, il registratore di cassa possiede altri componenti. Il lettore di codice a barre resta in attesa del passaggio di un prodotto da parte dell'operatore. La decodifica del codice a barre viene svolta da

un sistema centralizzato remoto, che opera come se fosse locale al registratore di cassa. Le informazioni che esso ritorna sono racchiuse all'interno del tipo `Item`. Il conto è rappresentato dal tipo `Bill`, che è un aggregato di oggetti di tipo `Item`. Il calcolo del totale avviene utilizzando le informazioni contenute nel conto, applicandovi eventuali sconti, calcolati direttamente dal registratore di cassa. Gli sconti vengono applicati utilizzando diverse strategie, a seconda della campagna di marketing attualmente in atto.

Si modelli il sistema sopra descritto, mediante un diagramma delle classi che includa opportuni design pattern. Utilizzando poi un diagramma di sequenza, si descrivano le azioni corrispondenti alla lettura del codice a barre di un prodotto e alla visualizzazione a video delle sue informazioni all'interno del conto aperto.

Soluzione

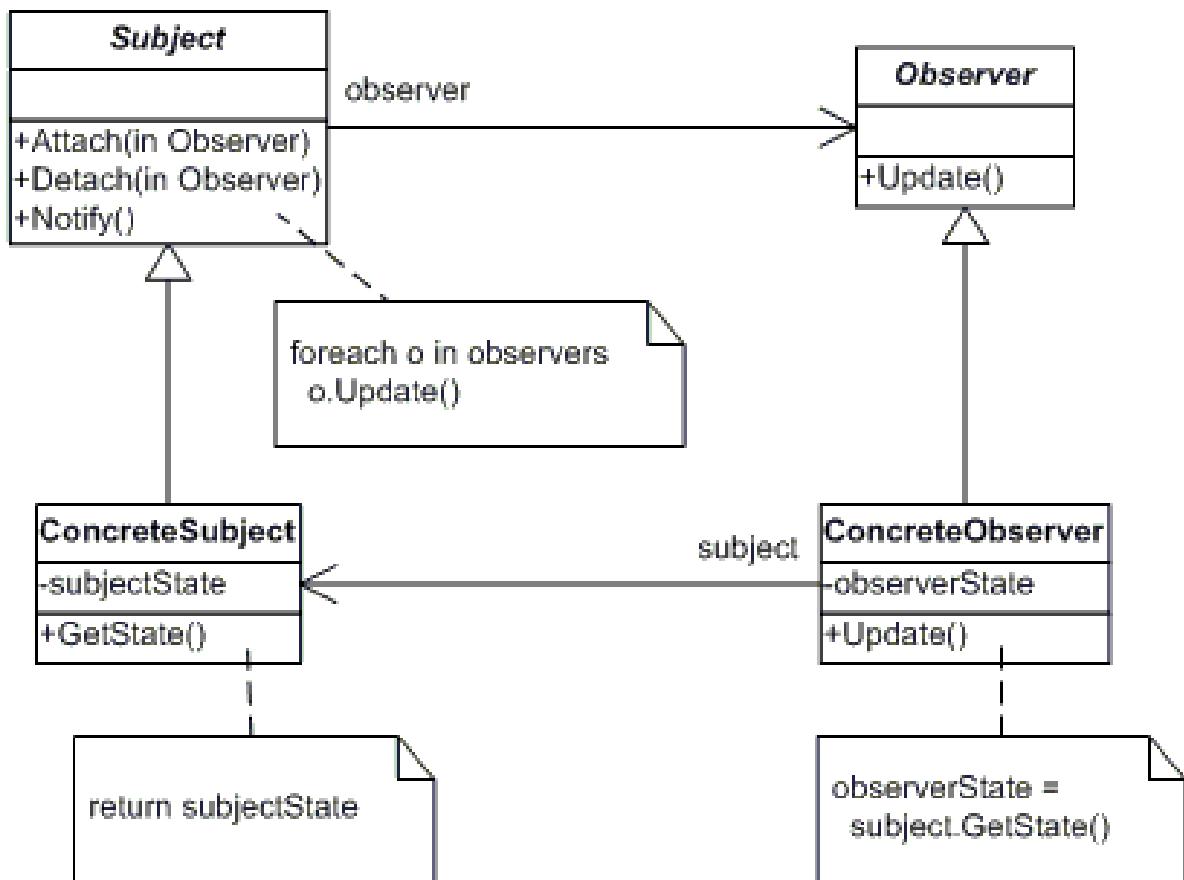
I pattern da applicare sono i seguenti: MVC, Observer, Proxy e Strategy pattern. Eventualmente è possibile individuare un Iterator pattern sul tipo `Bill`.



Esercizio 3 (3 punti)

Descrizione

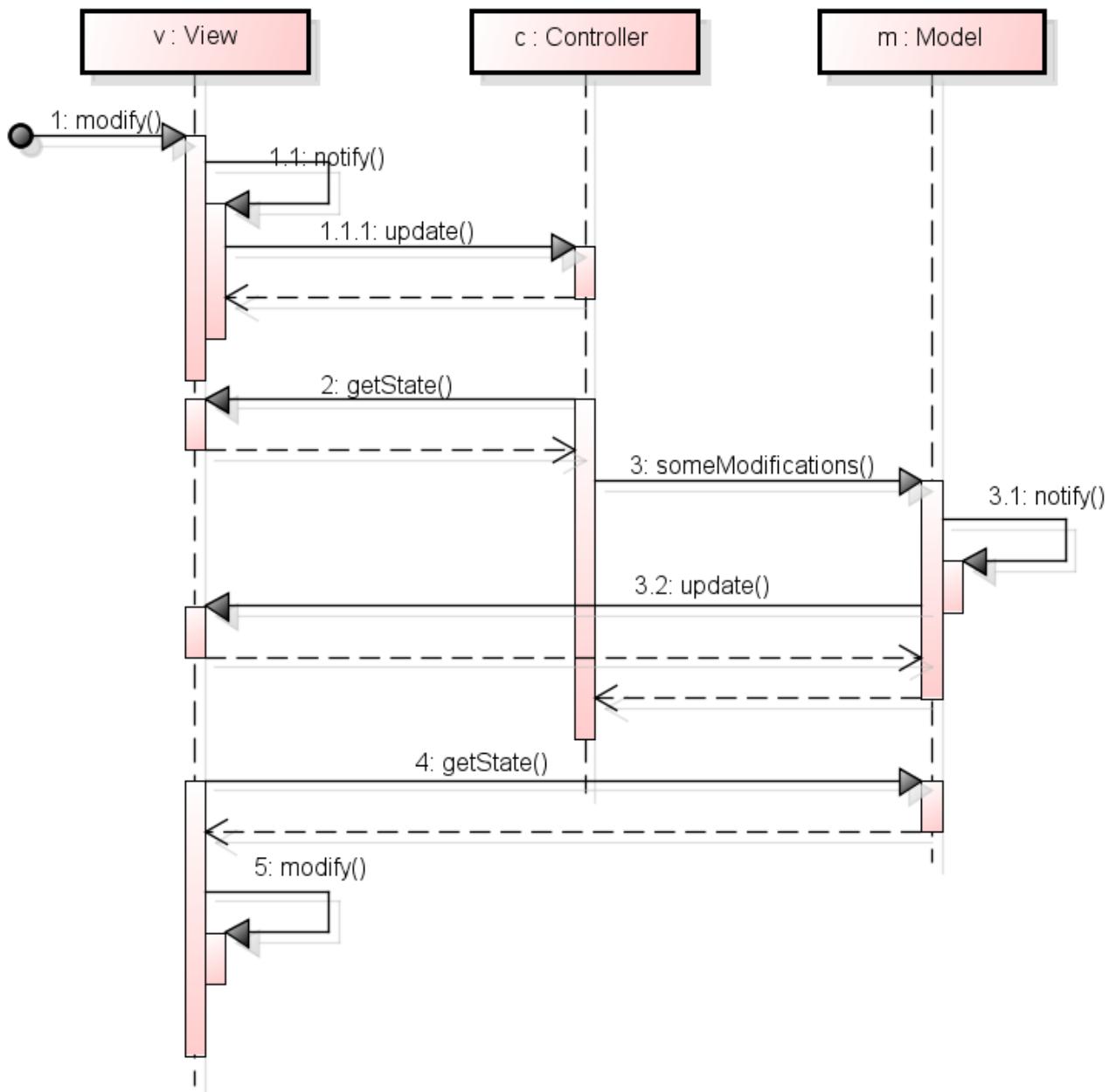
Nella sua versione base, nota come push model, il pattern Model View Controller, MVC, viene implementato utilizzando una doppia istanza del pattern Observer. Il diagramma delle classi qui di seguito rappresenta il pattern Observer.



Utilizzando un diagramma di sequenza, si modelli lo scambio di messaggi fra le componenti del pattern MVC che corrisponde al seguente flusso di operazioni: l'utente effettua un'operazione sulla vista, che modifica il modello e, di conseguenza, le informazioni visualizzate nella vista stessa.

Soluzione

Segue il diagramma di sequenza corrispondente.



Ingegneria del Software A.A. 2018/2019

Esame 2019-04-19

Esercizio 1 (6 punti)

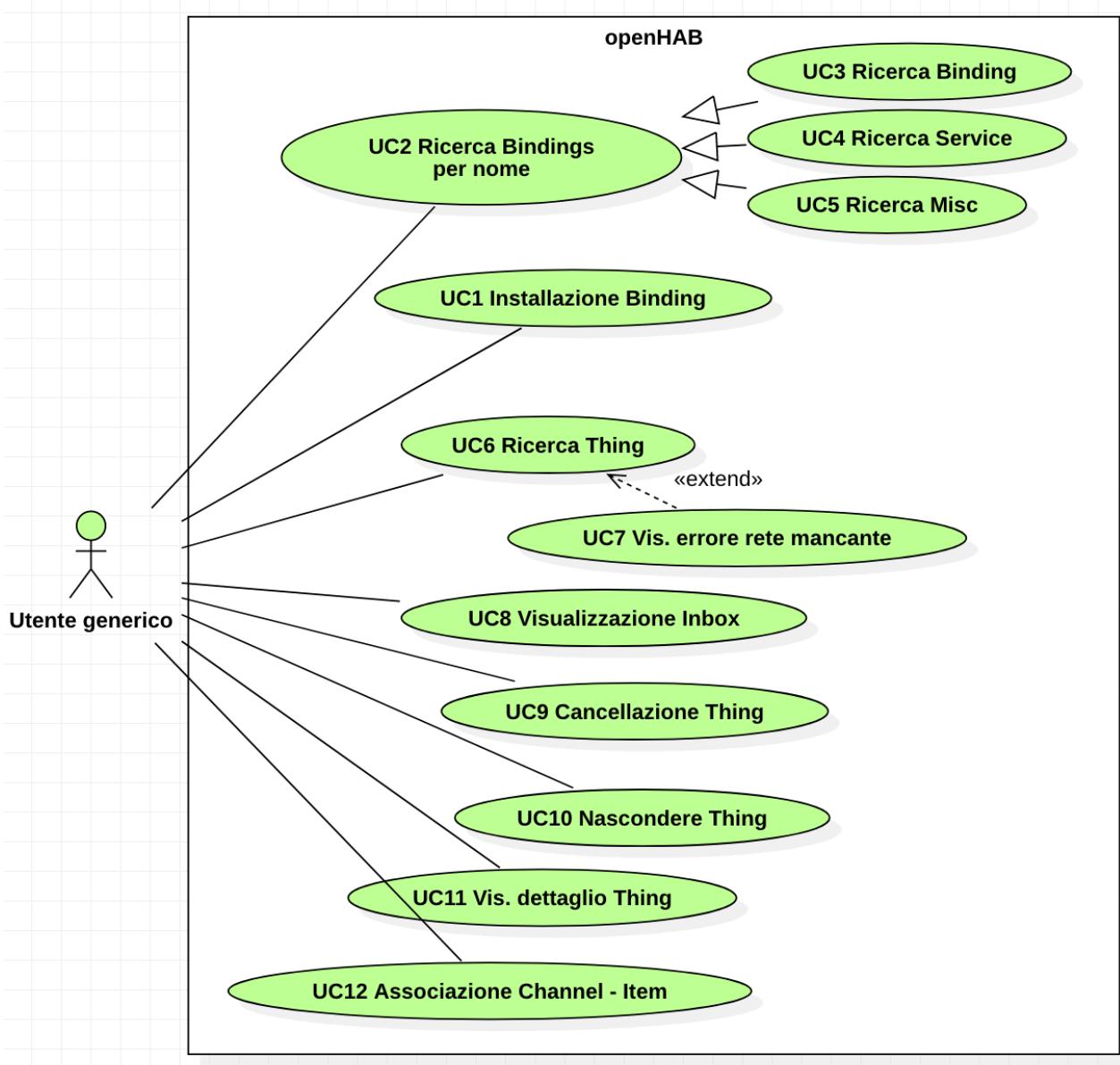
Descrizione

OpenHAB è un software open utilizzato per gestire in modo non dipendente dai *vendor* la domotica casalinga e non. Si basa sul concetto di astrazione. I dispositivi fisici vengono astratti all'interno del software come *Thing*. Le funzionalità esposte da ogni *Thing* sono dette *Channel*. Ad un *Channel* è possibile collegare un *Item*, che è la rappresentazione software della funzionalità associata ad un *Thing*. Il software di openHAB lavora a moduli. Attraverso un'interfaccia web è possibile aggiungere (installare) i cosiddetti *Binding*, ossia il software specifico per dialogare con i dispositivi di un particolare *vendor*. I *Binding* possono essere ricercati utilizzando un nome e sono suddivisi per categoria (*Bindings*, *Services*, *Misc*, ...). Una volta installato un *Binding* è possibile iniziare una ricerca di tutti i dispositivi collegati alla medesima rete locale del server openHAB. La ricerca visualizza tutti i *Thing* trovati in una lista, detta *Inbox*, oppure visualizza un errore se non è presente alcun collegamento alla rete locale. Nella lista è possibile visualizzare il nome del *Thing* ed una breve descrizione. Inoltre, è possibile cancellare l'oggetto oppure semplicemente nasconderlo. Nella pagina di dettaglio di un *Thing* è possibile visualizzare il suo stato (online/offline), una sua descrizione e la lista dei *Channel* ad esso associati. In questa pagina avviene l'associazione fra un *Channel* e un rispettivo nuovo *Item*.

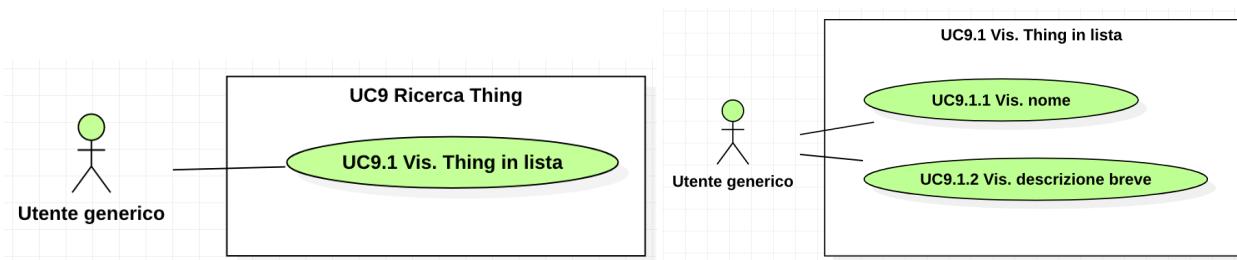
Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari sopra descritti. Non ne è richiesta la descrizione testuale.

Soluzione

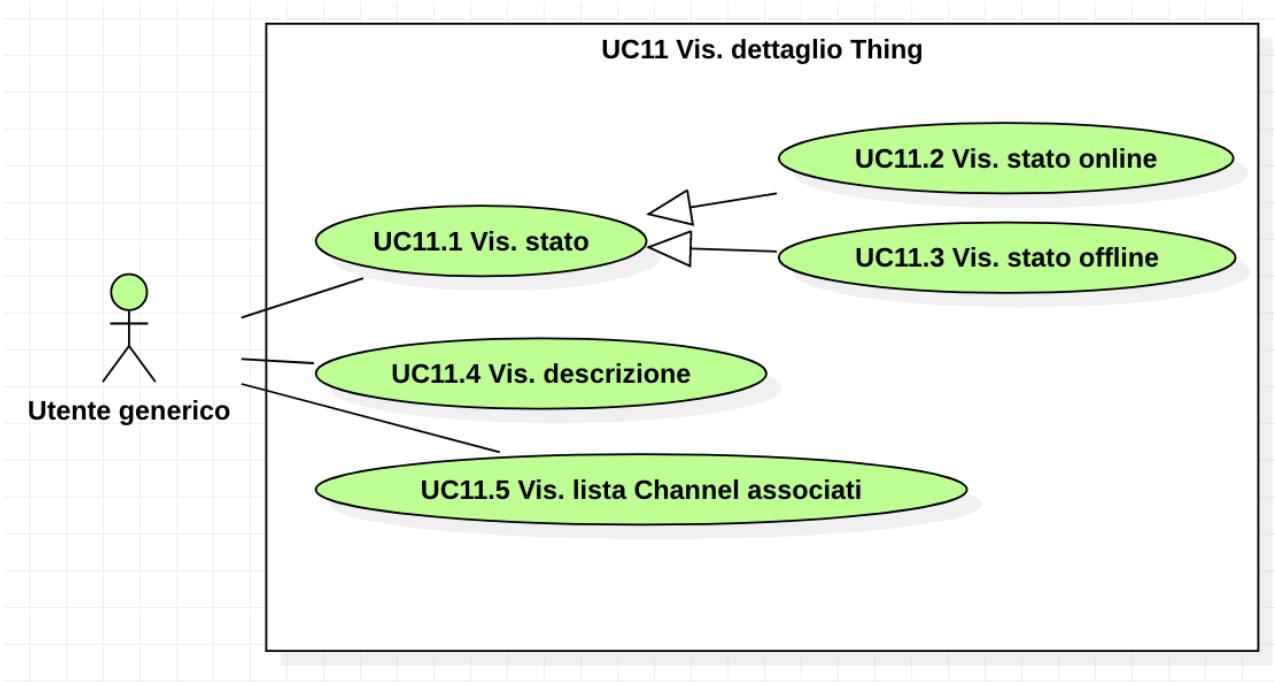
Il diagramma dei casi d'uso principale corrispondente è il seguente.



La visualizzazione dei Thing in lista è modellata come segue.



Infine, la visualizzazione del dettaglio di un Thing può essere modellata come segue.



Esercizio 2 (7 punti)

Descrizione

La rilevazione (*discovery*) di nuovi *Thing* in openHAB avviene con modalità *publish/subscribe*. Ogniqualvolta un nuovo dispositivo *hardware* per il quale è installato un *Binding* si registra nella medesima rete locale del *server openHAB*, esso lo rileva e crea un corrispondente *Thing* al suo interno. Il processo di rilevazione opera per *Binding*. Ogni *Thing* si interfaccia con una specifica istanza del *driver* messo a disposizione dal *Binding*. Il potere espressivo di openHAB sta proprio nell'uso del medesimo oggetto, *Thing*, per governare l'interazione con dispositivi diversi di *vendor* diversi, adattando di volta in volta solo lo specifico protocollo. Ogni *Thing* può essere di tipo *switch*, *roller shutter*, e *thermostat*. Ogni tipologia di *Thing* ha poi il proprio modo standard di adattare il *driver* fornito dai *Binding*.

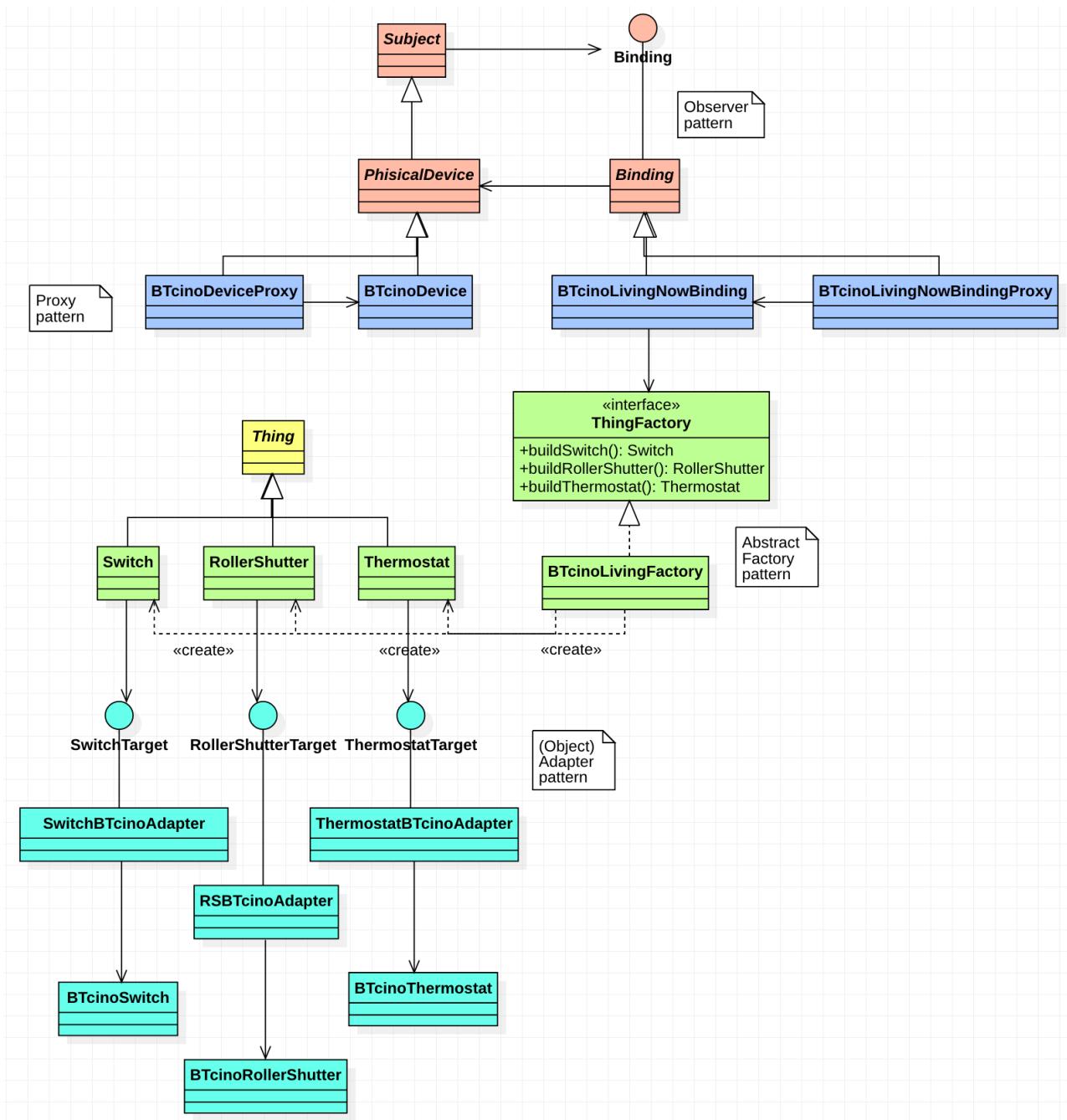
Si modelli tale sistema mediante un diagramma delle classi e i *design pattern* a esso pertinenti. Utilizzando un diagramma di sequenza, si descriva poi l'aggiunta di un nuovo *Thing* di tipo *switch* del *vendor* BTicino Living Now.

Soluzione

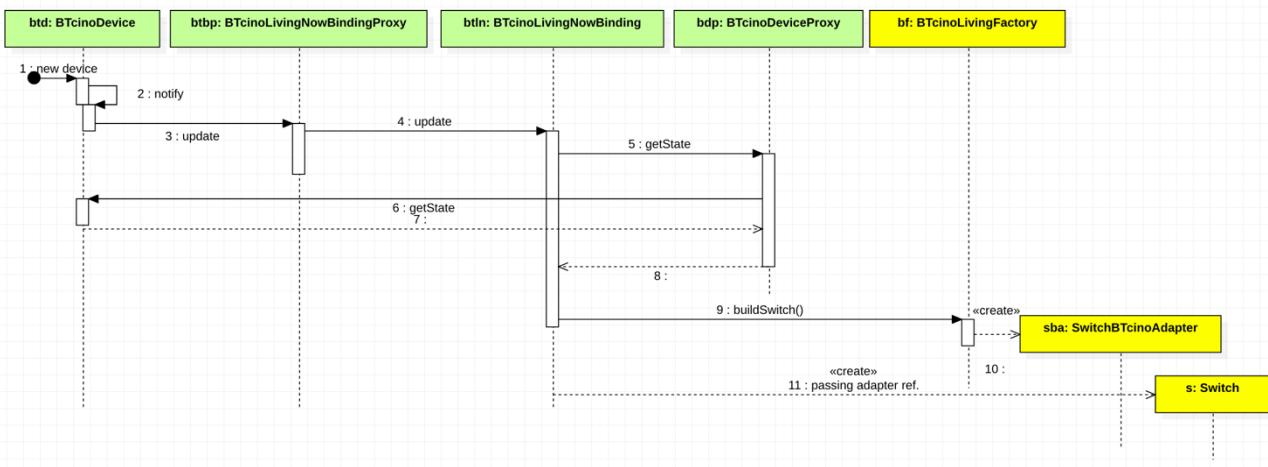
La soluzione prevede l'utilizzo dei seguenti design pattern:

- Observer
- Proxy
- Abstract factory
- Adapter (object)

Un possibile diagramma delle classi è il seguente.



Il diagramma di sequenza richiesto è invece il seguente.



Esercizio 3 (3 punti)

Descrizione

L'installazione di una lista di nuovi *Binding* all'interno del sistema opanHAB avviene utilizzando il seguente codice Java.

```

public class Binding {

    private final String bindingType;

    public static void install(Binding[] bindings) {
        for (Binding b: bindings) {
            switch (b.bindingType) {
                case "Bticino":
                    ((BTicinoBinding) b).installBTicino();
                    break;
                case "Hue":
                    ((PhilipsHueBinding b)).installHue();
                    break;
            }
        }
    }
}

public class BTicinoBinding {
    // Code that initializes super bindingType
    public void installBTicino() {

```

```

        System.out.println("Installing BTicino binding");
    }
}

public class PhilipsHueBinding {
    // Code that initializes super bindingType
    public void installHue() {
        System.out.println("Installing Philips Hue binding");
    }
}

```

Si modifichi il suddetto codice al fine di fare aderire il metodo `install` al principio *Open-Closed*.

Soluzione

È necessario uniformare i metodi nelle classi concrete in una unica interfaccia `install`, dichiarata in `Binding`. In questo modo, è possibile estendere il sistema aggiungendo nuove tipologie di *Binding*, senza tuttavia modificare il codice esistente.

```

public abstract class Binding {
    public abstract void install();
    public static void install(Binding[] bindings) {
        for (Binding b: bindings) {
            b.install();
        }
    }
}

public class BTicinoBinding extends Binding {
    @Override
    public void install() {
        System.out.println("Installing BTicino binding");
    }
}

public class PhilipsHueBinding extends Binding {
    @Override
    public void install() {
        System.out.println("Installing Philips Hue binding");
    }
}

```

}

Ingegneria del Software A.A. 2018/2019

Esame 2019-05-17

Esercizio 1 (6 punti)

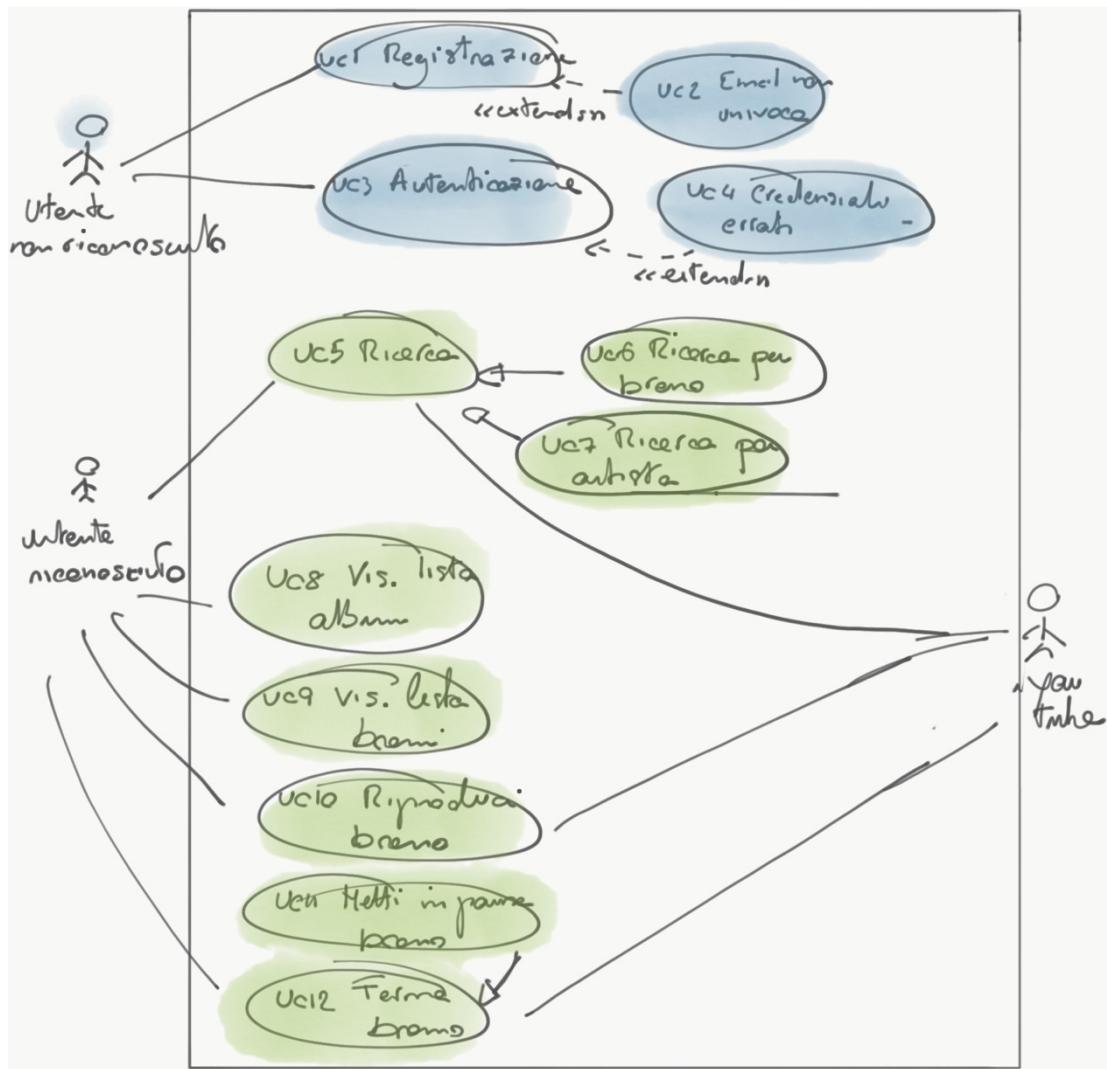
Descrizione

Netmus avrebbe dovuto essere il software per l'ascolto di brani musicali in streaming del futuro. Ideato nel 2010, venne velocemente soppiantato dall'arrivo di concorrenti quali Spotify e Google Music. Il servizio di streaming richiedeva dapprima una registrazione, tramite un'e-mail ed una password. L'e-mail doveva essere unica. La successiva fase di autenticazione avrebbe richiesto i medesimi dati. L'ascolto dei brani avveniva sfruttando la riproduzione dei video musicali su YouTube. Era possibile ricercare un brano sia per artista, sia per titolo del brano stesso. La prima ricerca ritornava una lista di album, la seconda i brani che corrispondevano al titolo inserito. Nel primo caso, di ogni album in lista erano visualizzati la copertina, il nome, e la durata complessiva. Le copertine erano recuperate all'occorrenza da un servizio esterno. Ogni brano della lista poteva essere riprodotto, messo in pausa o fermato. Inoltre, era possibile creare delle *playlist*, fornendo un nome ed aggiungendovi brani. Ogni playlist poteva essere condivisa su Facebook.

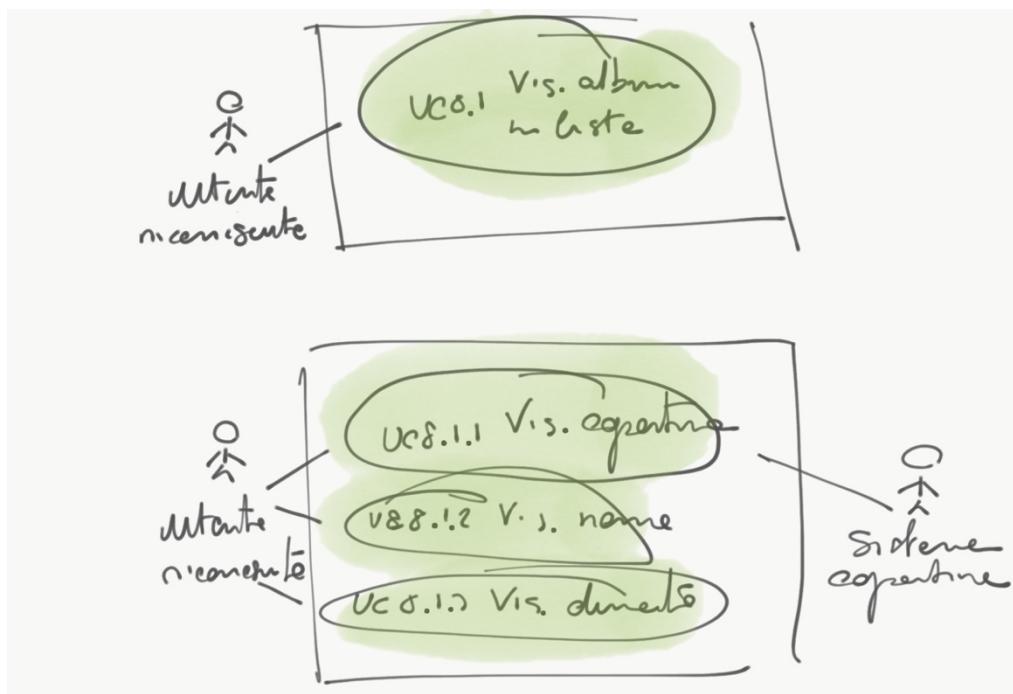
Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari sopra descritti. Non ne è richiesta la descrizione testuale.

Soluzione

Di seguito la soluzione.



In dettaglio, la visualizzazione delle informazioni di un album nella lista proveniente dalla ricerca è la seguente.



Esercizio 2 (7 punti)

Descrizione

Netmus era un sistema molto avanzato per il suo tempo dal punto di vista architetturale. Per limitare l'utilizzo di risorse, anticipando i tempi, faceva uso della *reactive programming*. In particolare, la riproduzione di un brano utilizzando YouTube avveniva attraverso un *listener* sulla disponibilità di nuove informazioni provenienti dal sito di streaming video. Ogni video veniva suddiviso in *chunk*, e la disponibilità di un nuovo *chunk* immediatamente notificata al riproduttore musicale. Questo, avvalendosi di un algoritmo di decodifica audio/video, separava i due canali, restituendo unicamente quello audio. L'algoritmo era in continuo miglioramento, quindi la struttura pensata per la decodifica permetteva la sua estensione in modo agevole. Per semplicità, un brano musicale era comunque visto come un file completo localmente, contenendo l'informazione audio, il titolo e l'immagine della copertina dell'album associato.

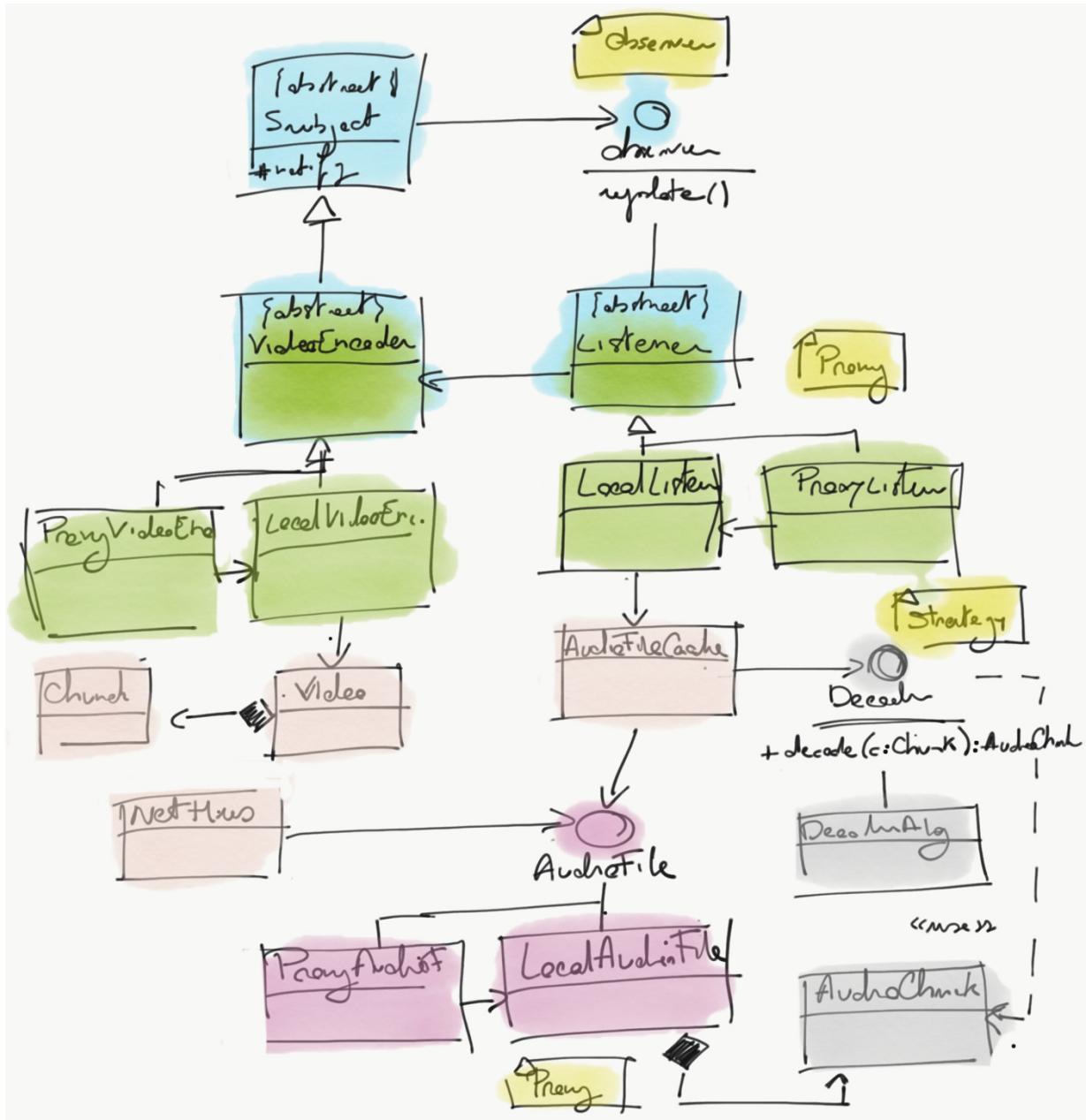
Si modelli tale sistema mediante un diagramma delle classi e i *design pattern* a esso pertinenti. Immaginando una richiesta di ascolto del brano "Bohemian Rhapsody" dei Queen, si utilizzi un diagramma di sequenza per descrivere l'interazione fra le componenti individuate.

Soluzione

I pattern individuati sono i seguenti:

- Observer
- Remote proxy
- Virtual proxy
- Strategy

Il diagramma delle classi è il seguente.



Il diagramma di sequenza è facilmente implementabile, di conseguenza.

Esercizio 3 (3 punti)

Descrizione

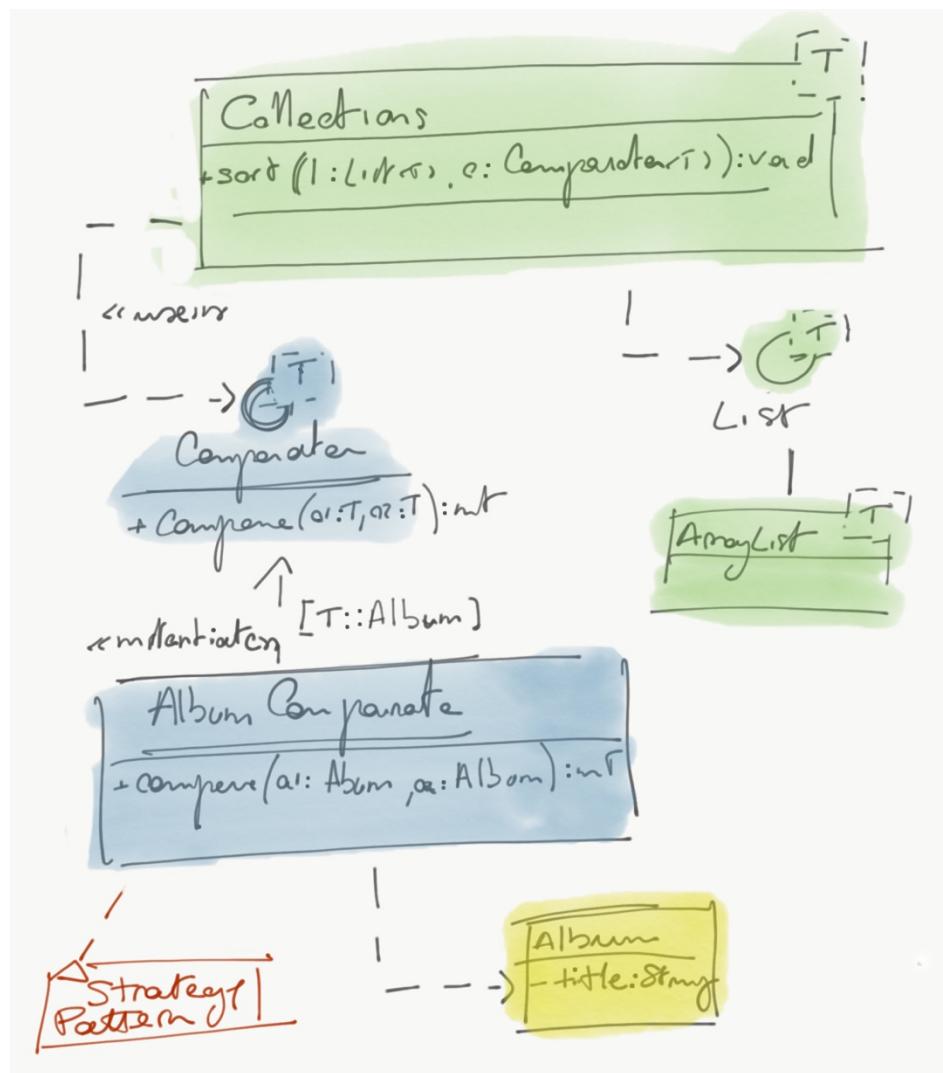
Il framework delle *Collection* in Java permette di ordinare una lista in modo arbitrario utilizzando il metodo della classe Collections, static <T> void sort(List<T> list, Comparator<? super T> c).

L'interfaccia Comparator contiene un unico metodo int compare(T o1, T o2).

Tale framework utilizza un noto *design pattern* della GoF. Si fornisca il diagramma delle classi che lo contestualizza, per una lista di oggetti della classe `Album`, utilizzando un comparatore che ordina per `title`.

Soluzione

Il pattern richiesto è lo Strategy pattern e il diagramma delle classi relativo all'esercizio è il seguente.



Ingegneria del Software A.A. 2018/2019

Esame 2019-06-17

Esercizio 1 (6 punti)

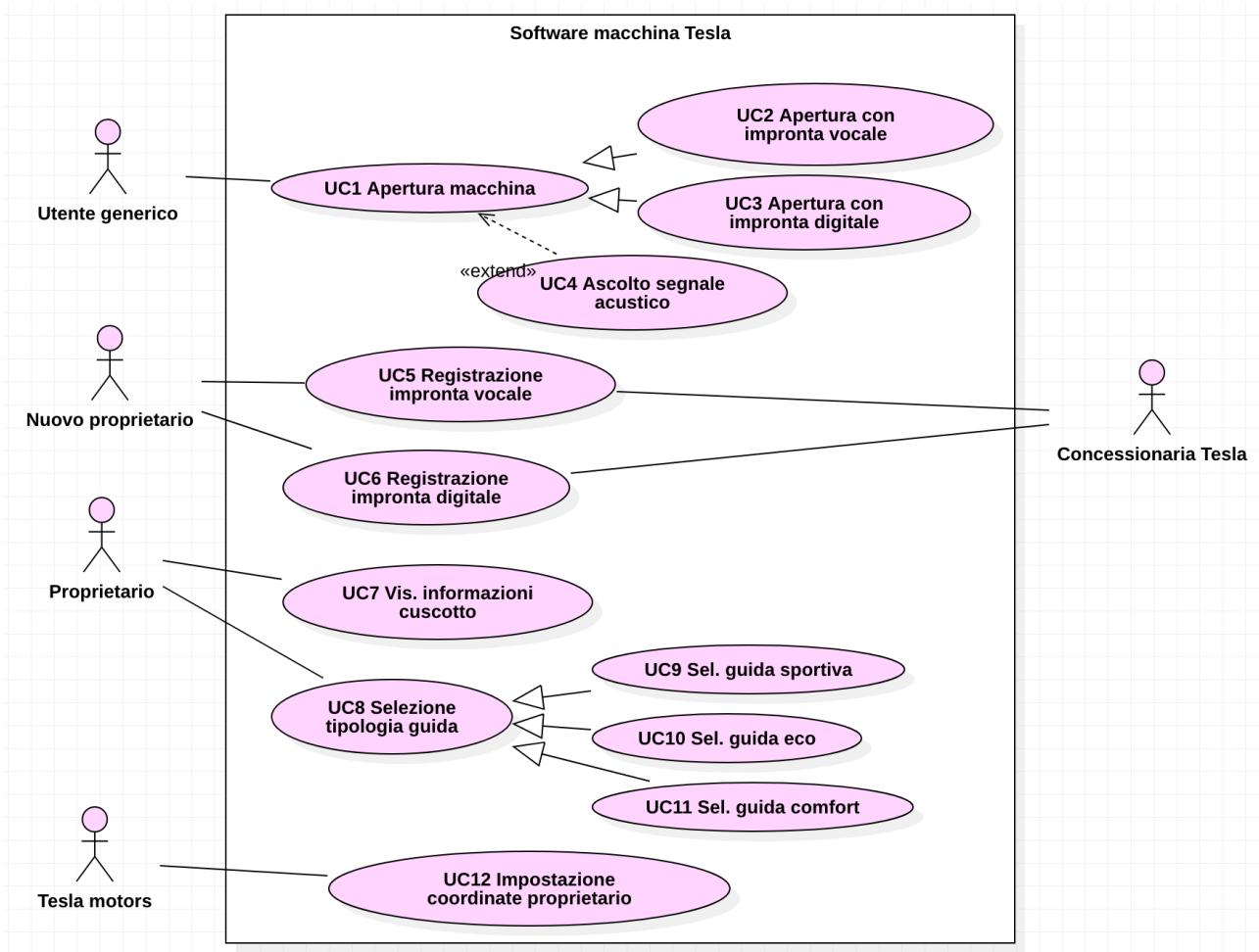
Descrizione

Fin dalla sua nascita nel 2003, Tesla Motors si è caratterizzata come grande innovatore nel campo delle auto elettriche. Il fiore all'occhiello della sua produzione sono le componenti elettroniche e le applicazioni software installate sulle proprie vetture. La tipica auto Tesla può essere aperta dal proprietario utilizzando un'impronta vocale o digitale. Al primo utilizzo in concessionaria, entrambe tali impronte vengono registrate nel software della vettura, in modo da garantirne il futuro riconoscimento. Un successivo tentativo di accesso non riuscito causa l'emissione di un segnale acustico di allarme. Una volta all'interno, il cruscotto della vettura visualizza una serie di informazioni tra le quali: lo stato delle batterie (caricate, da caricare, in carica), la data e l'ora del giorno, e una diagnostica del motore. Per ogni voce della lista delle componenti del motore, il display riporta il nome, e lo stato di usura. Interagendo con l'immagine del motore è possibile selezionare la tipologia di guida desiderata, a scelta tra sportiva, comfort, eco. Utilizzando una *app* dedicata per *smartphone*, è inoltre possibile fornire le proprie coordinate posizionali alla vettura, per farla guidare da sola verso la posizione del proprietario.

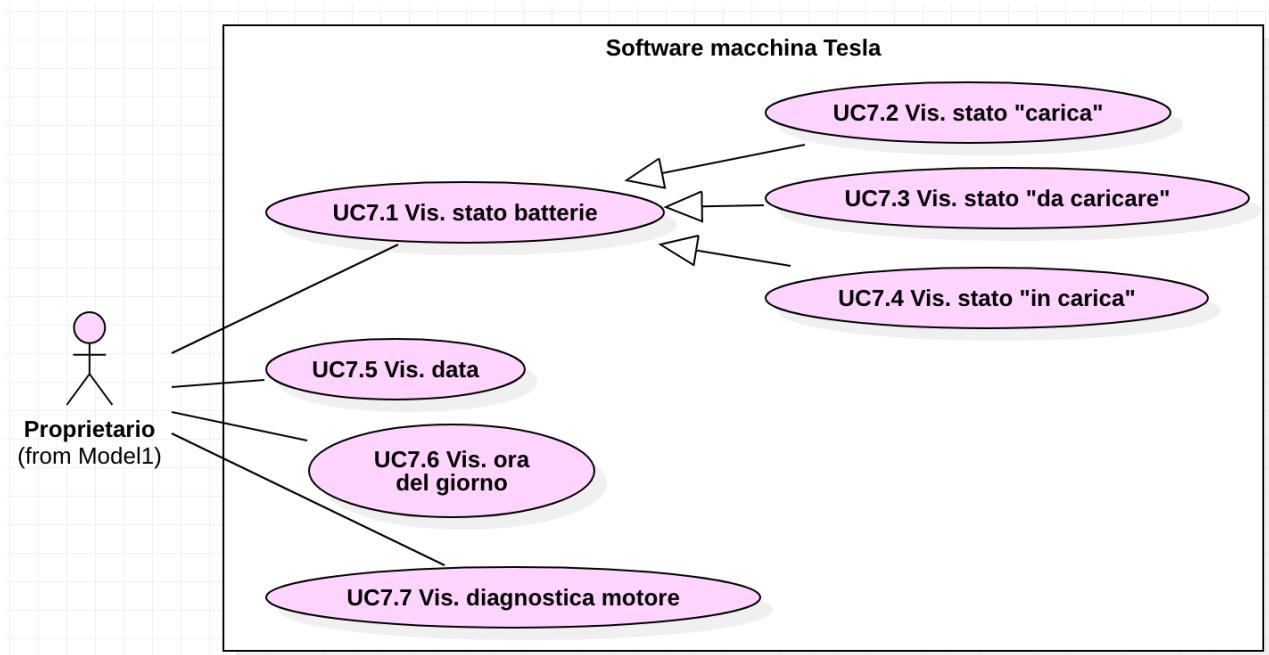
Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari sopra descritti. Non ne è richiesta la descrizione testuale.

Soluzione

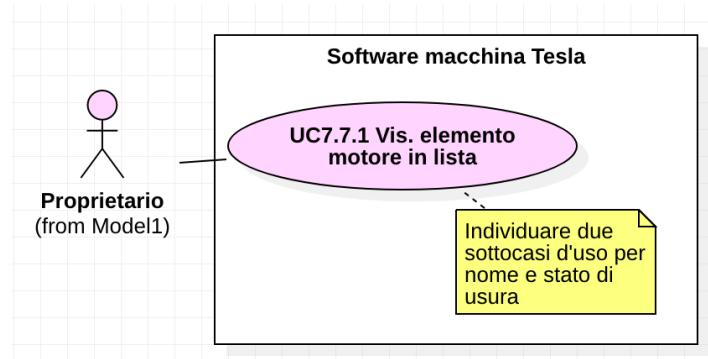
Una possibile soluzione è la seguente. Inizialmente è necessario analizzare il software installato all'interno della macchina. Si noti che si sono individuati tre differenti attori principali relativi al proprietario della macchina. Il sistema auto, infatti, si trova in tre stati completamente differenti durante le interazioni con ognuno di questi attori.



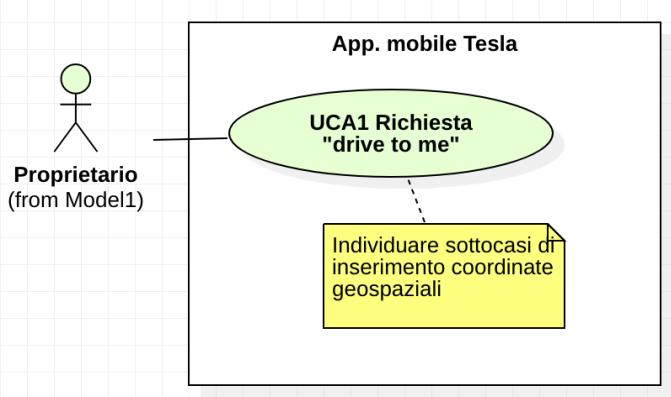
La visualizzazione del cruscotto richiede l'analisi di ulteriori sotto casi d'uso.



E per completare, il diagramma seguente.



L'esercizio richiede anche l'analisi di una funzionalità di un'applicazione mobile esterna alla macchina.



Esercizio 2 (7 punti)

Descrizione

La guida autonoma di Tesla è possibile perché una grande quantità di sensori è installata sulla vettura per fornire al *software* di controllo l'informazione necessaria per conoscere costantemente lo stato dei suoi apparati e dell'ambiente circostante. In particolare, le molte telecamere poste sulla vettura producono una immagine (*frame*) ogni 10 millisecondi. Tale flusso di immagini viene catturato da opportuni *listener*, che successivamente le forniscono a un algoritmo di apprendimento automatico, capace di categorizzare le cose o le persone rilevate intorno alla vettura. Qualora venga rilevato un pedone entro 20 metri dalla traiettoria attuale della vettura, tale informazione viene notificata a una centralina di controllo che attiva immediatamente i sistemi di sicurezza attiva, i quali arrestano la vettura. La centralina è in grado di eseguire diverse operazioni su molteplici componenti dell'auto, pur mantenendo un'unica interfaccia di esecuzione, ciò che garantisce massima estensibilità del *software*.

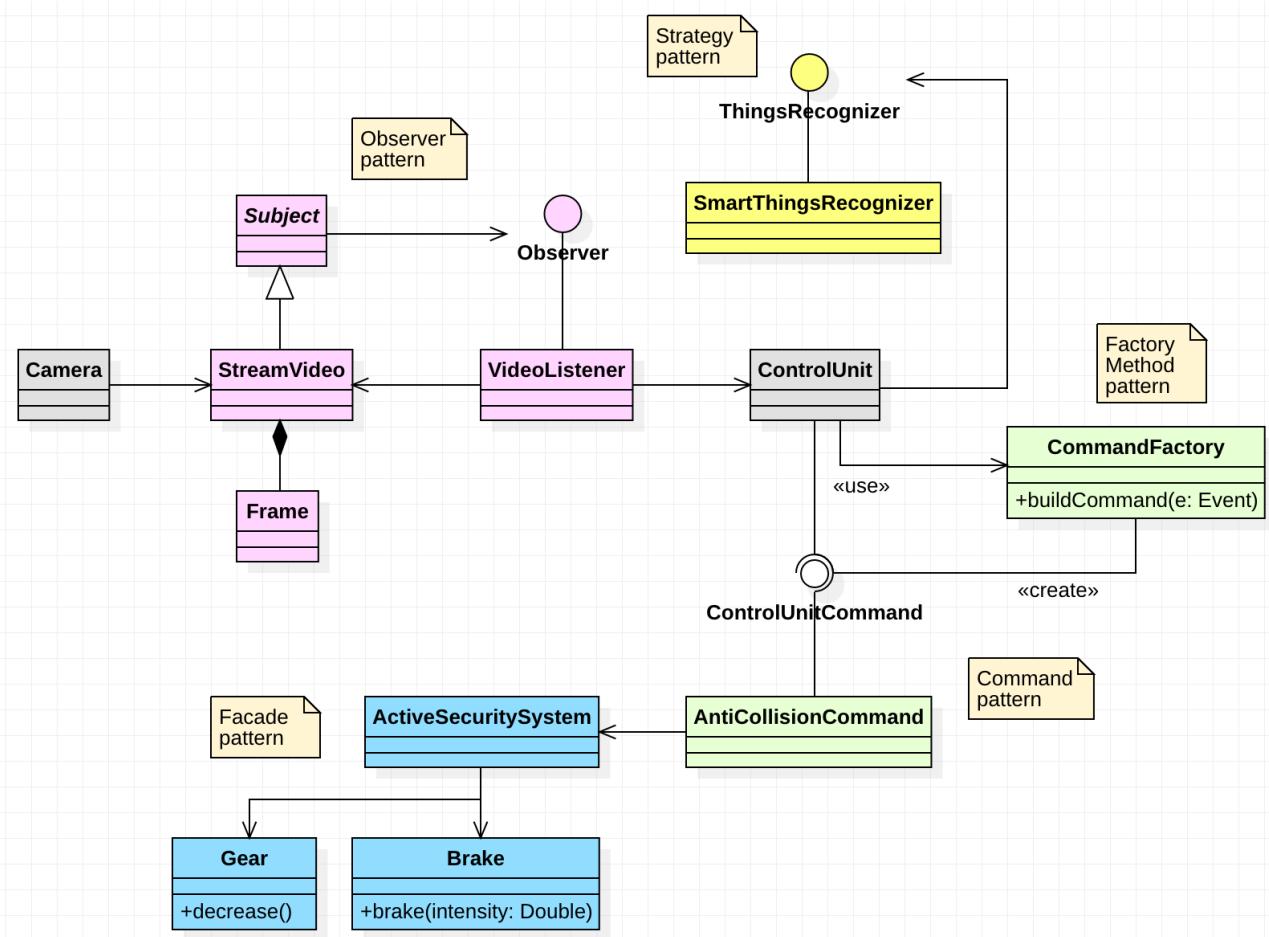
Si modelli il sistema sopra descritto mediante un diagramma delle classi e i *design pattern* a esso pertinenti. Immaginando che un pedone venga rilevato in rotta di collisione con la vettura, si utilizzi un diagramma di sequenza per descrivere l'interazione fra le componenti individuate, fino al completo arresto della vettura.

Soluzione

La soluzione prevede l'utilizzo dei seguenti *design pattern*:

- Observer pattern, per acquisire le informazioni dalle telecamere dall'auto
- Strategy pattern, per applicare l'algoritmo di riconoscimento degli ostacoli

- Command pattern, per gestire in modo standard le operazioni all'interno della centralina di controllo
- Factory method pattern, per costruire i comandi da eseguire sulla base degli eventi scatenati nella centralina
- Facade pattern, per gestire tutte le operazioni necessarie sulle varie componenti della macchina per implementare la sicurezza attiva



Il diagramma di sequenza associato ne deriva in modo lineare.

Esercizio 3 (3 punti)

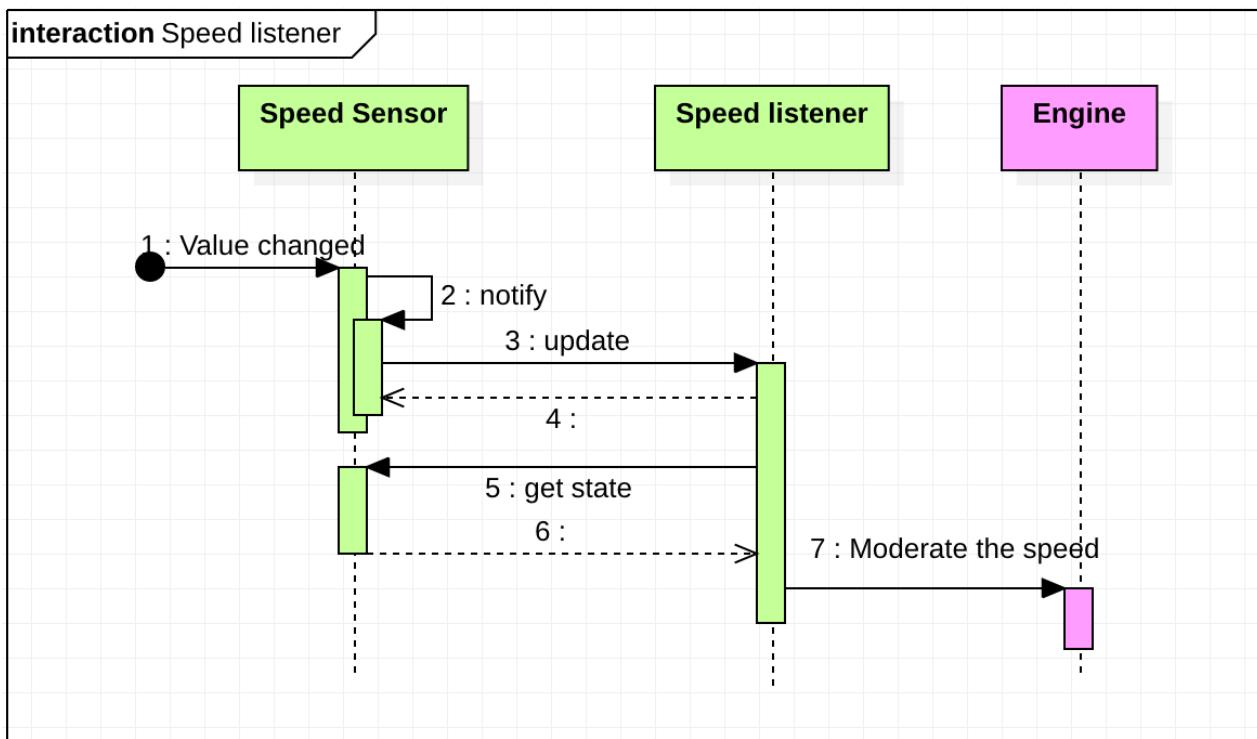
Descrizione

Poiché il possessore di una vettura Tesla si aspetta elevate prestazioni da tutti gli apparati, meccanici ed elettronici, della vettura, il *software* installato in essa utilizza ampiamente la *reactive programming*. Basata sul concetto di *observer*, questo paradigma di programmazione permette di reagire prontamente a eventi esterni. Per questo motivo, il *reactive programming* è utilizzato proficuamente, per esempio, per gestire i sensori che rilevano la velocità della macchina.

Riferendosi al pattern *Observer*, fornire il diagramma di sequenza di una possibile interazione fra i sensori di velocità e un *listener* che ne utilizza le informazioni.

Soluzione

Per semplicità, si suppongano unicamente messaggi sincroni. I tipi che partecipano al *pattern Observer* sono Speed Sensor e Speed Listener, rispettivamente in qualità di *subject* e *observer*.



Ingegneria del Software A.A. 2018/2019

Esame 2019-07-15

Esercizio 1 (6 punti)

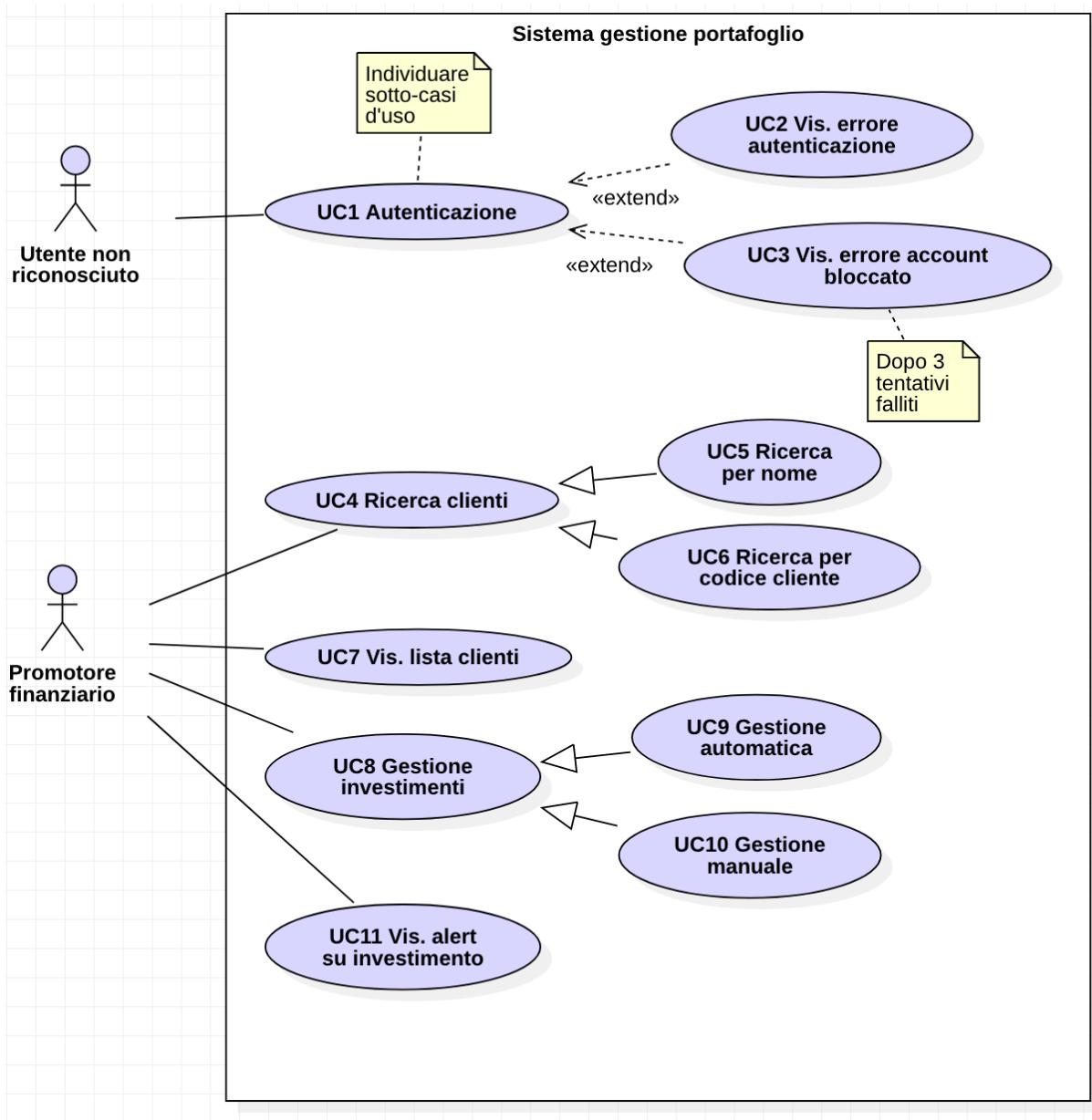
Descrizione

La piattaforma di investimenti della banca *SmartBank* permette ai promotori finanziari di proporre ai propri clienti soluzioni diversificate e altamente personalizzabili. La piattaforma prevede autenticazione tramite numero unico di matricola (codice cliente) e *password* associata. L'errore nell'inserimento della *password* porta alla visualizzazione di un errore. Al terzo errore di immissione *password*, il sistema blocca l'utente, impedendogli di autenticarsi. Ogni promotore può effettuare ricerche all'interno del proprio portafoglio clienti, per nome o codice cliente. Per ciascun cliente, la lista visualizza il nominativo (nome e cognome), il profilo di rischio associato (alto/medio/basso), e il capitale totale gestito dalla banca. Il dettaglio della posizione del cliente permette la gestione dell'insieme di fondi associati agli investimenti. Il promotore può scegliere fra due tipologie di investimenti: manuale, e automatico. Nel caso dell'investimento manuale, il promotore deve inserire il nome dell'azione da comprare, scegliendola fra quelle disponibili, e l'importo massimo da impiegare. Nel caso dell'investimento automatico, il promotore deve solo specificare quanto il cliente possa rischiare, mentre il sistema opera in autonomia comprando o liquidando diverse quantità di azioni da diversi fondi, fino al raggiungimento del *plafond* corrispondente al livello di rischio fissato. Qualora il capitale di un cliente sia stato intaccato per più del 30% rispetto all'investimento iniziale, il sistema genera un *alert*, visualizzabile all'interno dell'applicazione.

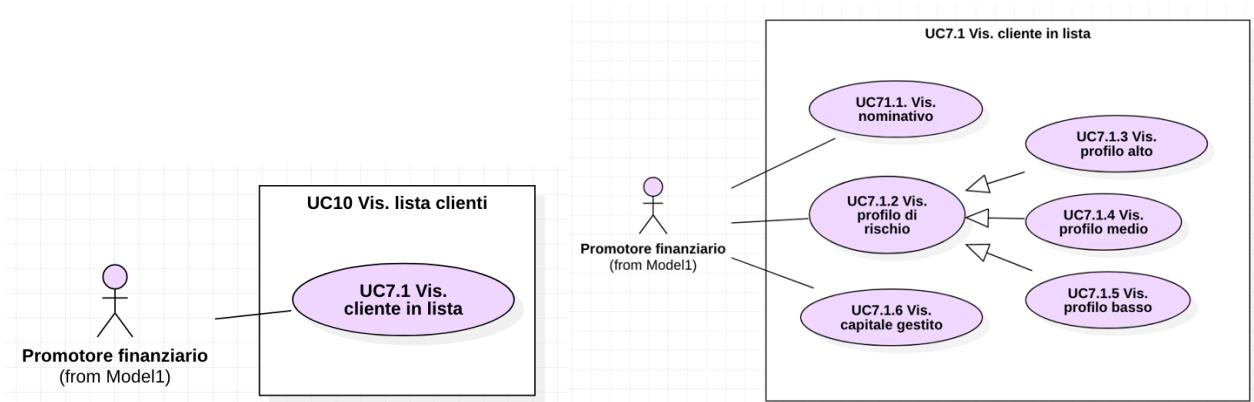
Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari sopra descritti. Non ne è richiesta la descrizione testuale.

Soluzione

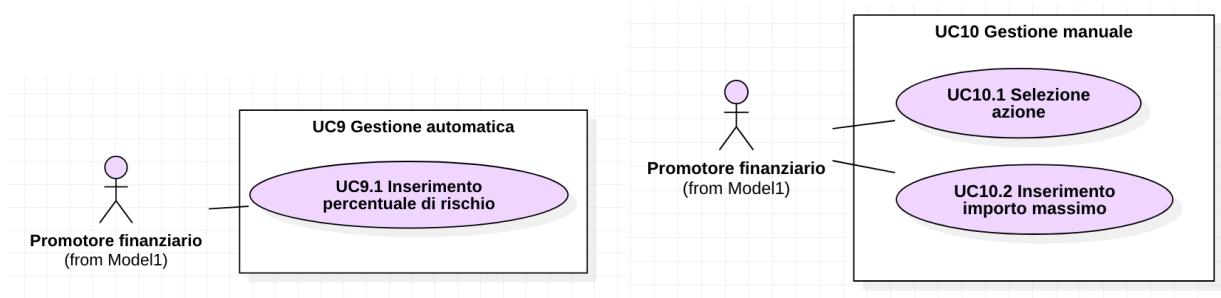
Un possibile diagramma dei casi d'uso associato al suddetto testo è il seguente.



La gestione della visualizzazione delle informazioni dei singoli clienti è modellata dai seguenti casi d'uso.



Infine la gestione degli investimenti può essere modellata come segue.



Esercizio 2 (7 punti)

Descrizione

L'applicazione per gli investimenti della *SmartBank* è tecnologicamente all'avanguardia. Il motore di investimento tiene monitorate costantemente diverse fonti di informazione digitali esterne al sistema, come ad esempio Twitter. Restando in ascolto sui *tweet* di diversi profili influenti sul mercato finanziario, il sistema valuta ogni nuovo post pubblicato. Per questioni di proprietà intellettuale, il sistema non può usare le API fornite da Twitter. Il testo recuperato dalla fonte Twitter viene trasformato in *token* (o *term*). Successivamente, usando un algoritmo di apprendimento automatico, il sistema calcola una misura di quanto il *tweet* possa influire negativamente o positivamente su un determinato titolo azionario. Tale algoritmo è alla base di tutta la piattaforma e pertanto nuove versioni migliorative di esso vengono rilasciate regolarmente. Sulla base della misura di influenza determinata dall'algoritmo, il sistema scorre il portafoglio di azioni di ogni singolo cliente, utilizzando un criterio di attraversamento basato sul valore delle azioni. Le azioni vengono quindi vendute o comprate di conseguenza.

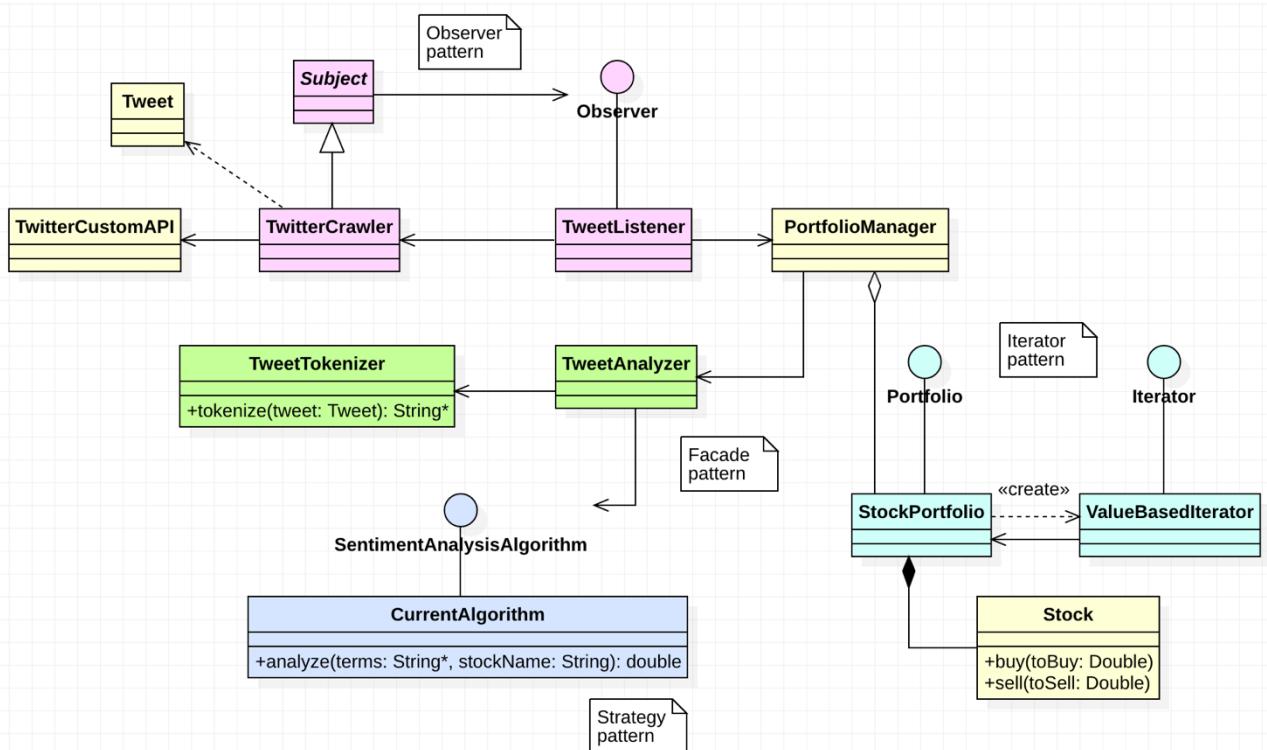
Si modelli il sistema sopra descritto mediante un diagramma delle classi e i *design pattern* a esso pertinenti. Immaginando un nuovo *tweet* sul profilo della CNN, si utilizzi un diagramma di sequenza per descrivere l'interazione fra le componenti individuate, immaginando una vendita di azioni Apple e un acquisto di azioni Google all'interno di un portafoglio cliente.

Soluzione

Una possibile soluzione prevede l'utilizzo dei seguenti *design pattern*:

- Observer pattern
- Strategy pattern
- Iterator pattern
- Facade pattern

Il diagramma delle classi corrispondente è il seguente.



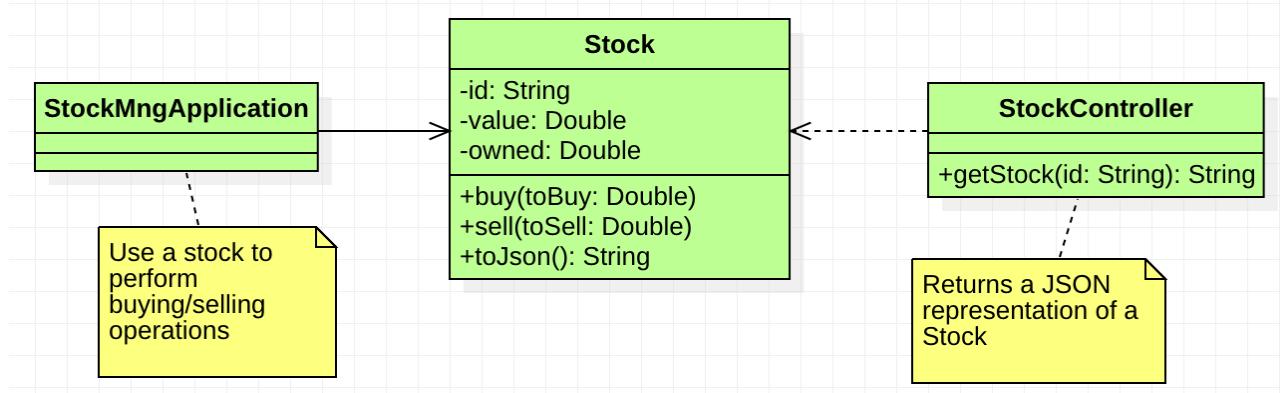
Il diagramma di sequenza deriva linearmente dal suddetto diagramma.

Esercizio 3 (3 punti)

Descrizione

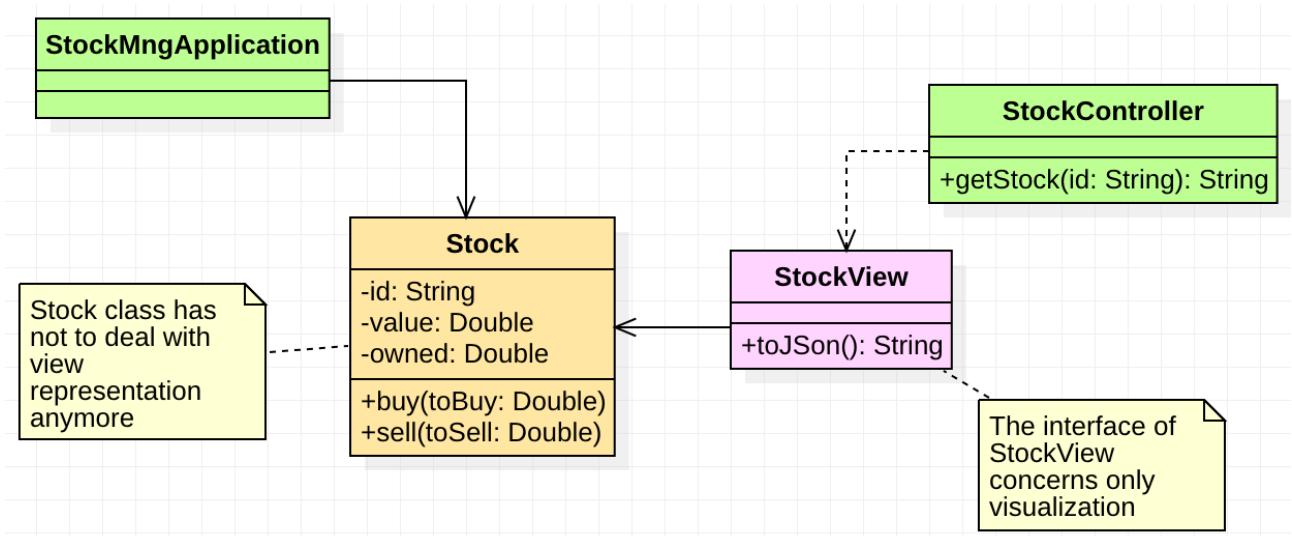
La classe **Stock** è utilizzata per gestire oggetti che rappresentano azioni finanziarie, sia come parte del processo di compravendita, che per esporre tali informazioni in formato JSON tramite un servizio REST.

Si modifichi il diagramma delle classi riportato in figura in modo da farlo aderire al “Single Responsibility Principle”.



Soluzione

L'applicazione del “Single Responsibility Principle” permette di isolare le dipendenze rispetto ai clienti della classe **Stock**, che deve essere pertanto suddivisa in due tipi distinti.



Ingegneria del Software A.A. 2018/2019

Esame 2019-08-19

Esercizio 1 (6 punti)

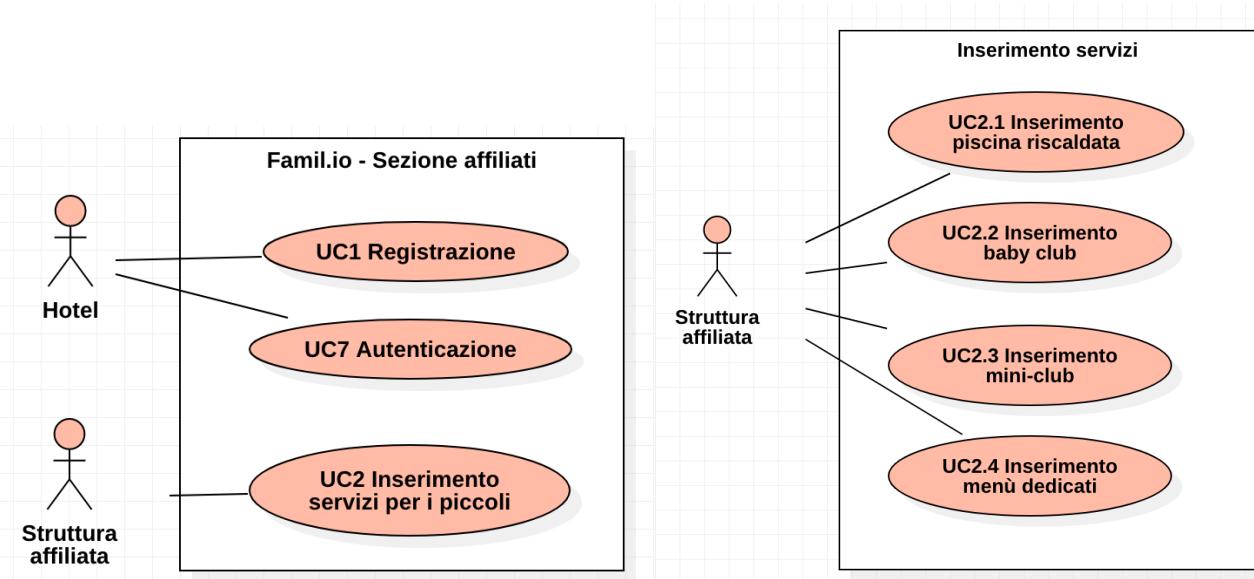
Descrizione

Il portale *Famil.io* è dedicato alla prenotazione di vacanze in strutture che offrono servizi dedicati anche ai più piccoli. Sul portale, le strutture interessate possono affiliarsi registrandosi, e successivamente inserendo i servizi offerti per i più piccoli, quali per esempio: piscina riscaldata, *baby* o *mini-club*, menù dedicati. I visitatori del portale possono effettuare ricerche inserendo contemporaneamente le seguenti informazioni: località della struttura, numero di stelle (da 1 a 5), presenza SPA e centro benessere, presenza piscina riscaldata, mini-club e menù dedicati ai più piccoli. I risultati della ricerca sono visualizzati in una lista che, per ogni struttura, riporta una sua foto, una brevissima descrizione, e una serie di icone che identificano i servizi offerti. Accendendo al dettaglio della struttura, invece, è possibile richiedere la disponibilità di camere inserendo le date di arrivo e partenza, il numero di adulti, e il numero di bambini. Infine, il portale permette anche di effettuare il versamento dell'eventuale caparra, utilizzando il servizio esterno *Paypal*.

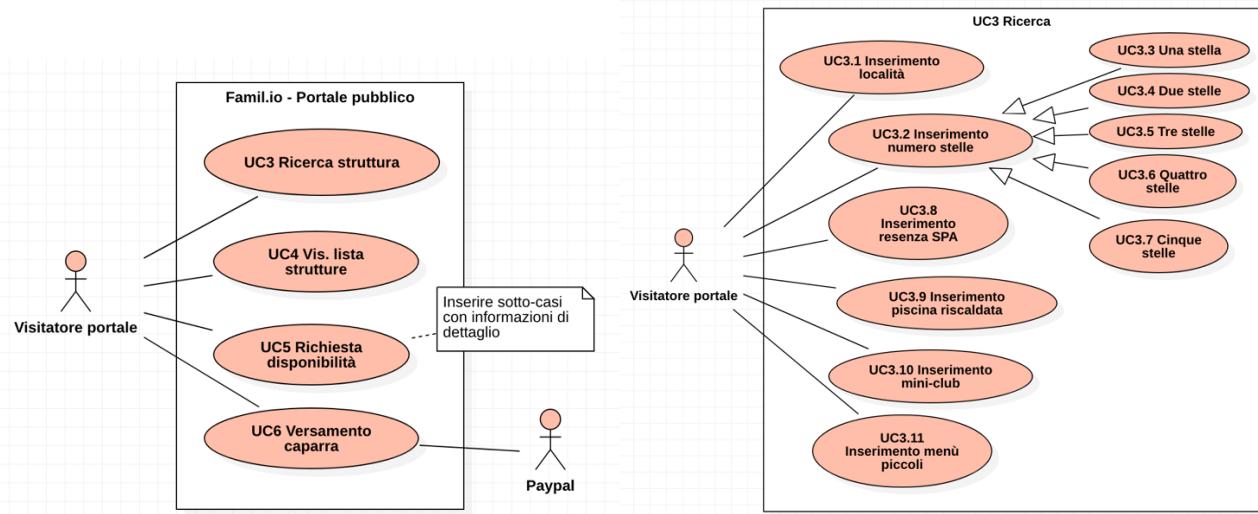
Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari sopra descritti. Non ne è richiesta la descrizione testuale.

Soluzione

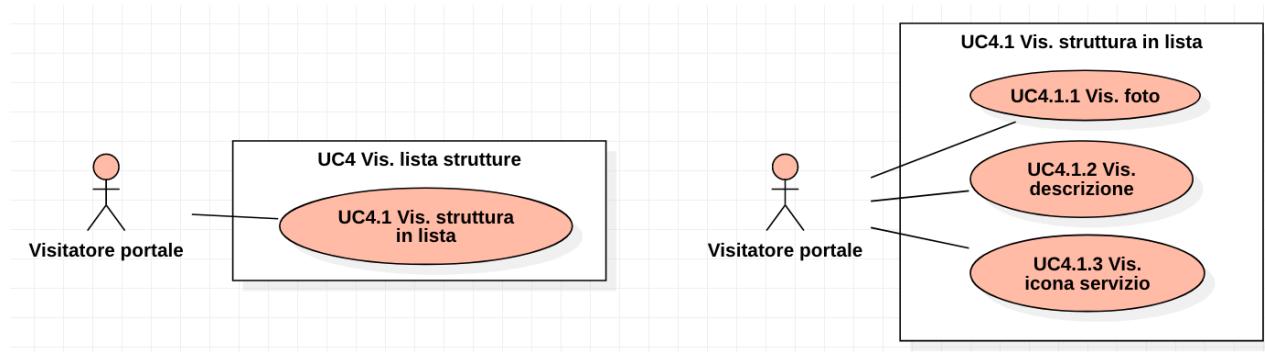
Una possibile soluzione è la seguente. Prima di tutto, si modellino i casi d'uso collegati alle "strutture convenzionate".



Successivamente, si modellino i casi d'uso collegati agli ospiti, che non hanno necessità di alcuna registrazione.



La visione dei risultati della ricerca può essere modellata come segue.



Esercizio 2 (7 punti)

Descrizione

Famil.io è un portale all'avanguardia. Tra le sue funzionalità avanzate, esso include il tracciamento automatico, in tempo reale, delle navigazioni e delle azioni degli utenti sul portale. Una sonda *Javascript* direttamente immersa nel portale visualizzato dagli utenti, traccia costantemente le loro azioni. Per ottimizzare il traffico da e verso il portale, la sonda salva i dati all'interno di una *cache* locale. Quando essa è piena, la sonda ne riversa il contenuto su un *server* centrale, che è sempre in suo ascolto. I dati raccolti vengono da esso inoltrati a un algoritmo di apprendimento automatico, costituito da una componente *trainer* e una componente *classifier*. La prima aggiorna il modello di apprendimento relativo a ogni singolo utente. La seconda fornisce una categorizzazione dell'utente secondo il corrispondente modello. Sulla base di tale categorizzazione, il portale crea pubblicità mirata allo specifico profilo di quell'utente, che viene inoltrata tramite posta elettronica e *social media* quali Facebook e Twitter. Per facilitare l'aggiunta di nuove modalità di inoltro e condivisione, l'utilizzo di tali canali di comunicazione è reso standard, utilizzando un'interfaccia unica di esecuzione.

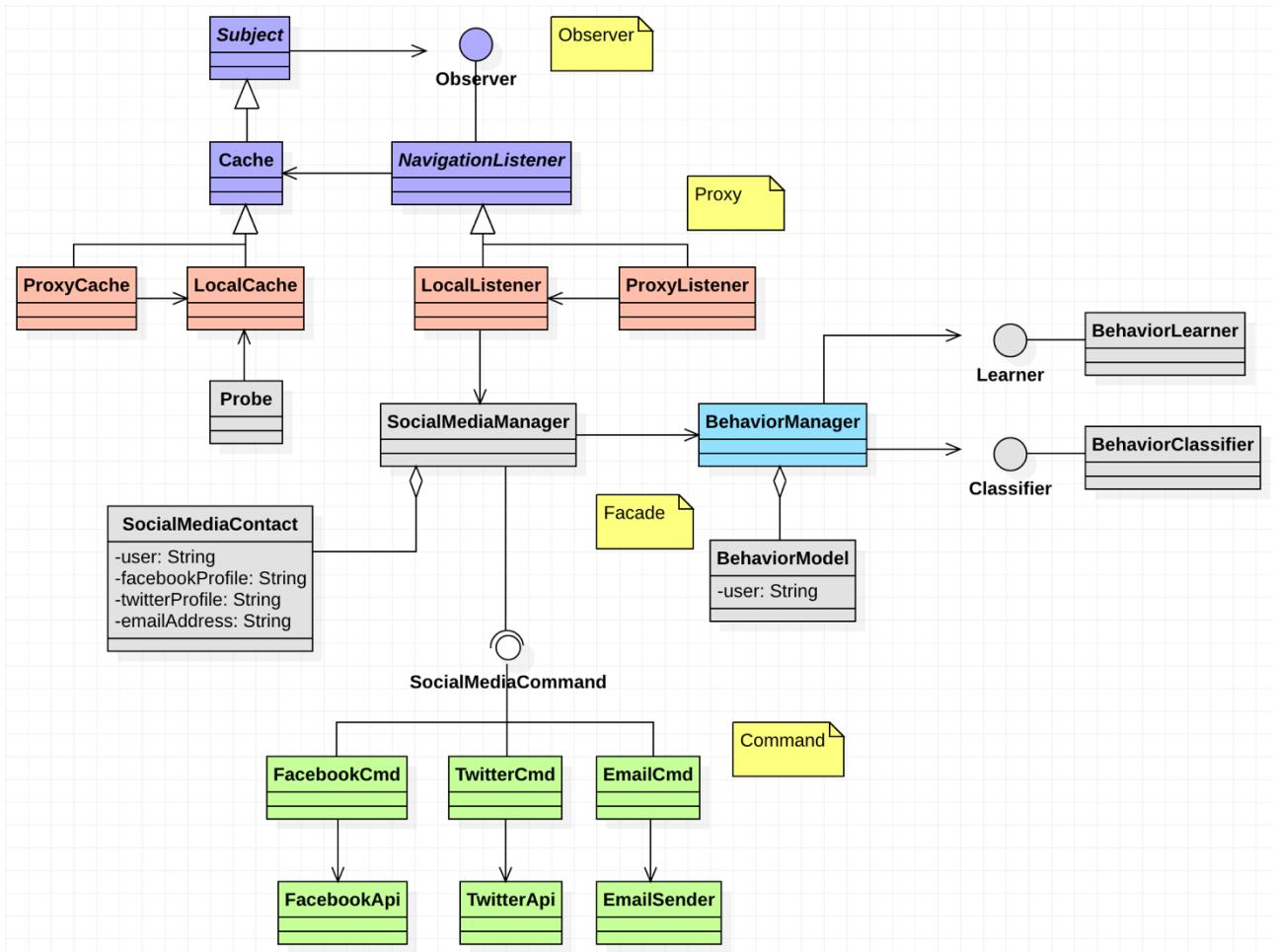
Si modelli il sistema sopra descritto mediante un diagramma delle classi e i *design pattern* a esso pertinenti. Si modelli inoltre, utilizzando un diagramma di sequenza, l'arrivo di un nuovo pacchetto di informazioni per l'utente "Riccardo Cardin", che può essere contattato utilizzando il *social-media* Facebook e il canale e-mail

Soluzione

La soluzione individuata prevede l'utilizzo dei seguenti *design pattern*:

- Observer
- Proxy
- Facade
- Command

Il diagramma delle classi che modella una possibile soluzione è il seguente.



In particolare, essendo in esecuzione su *host* differenti, la cache ed il sistema di gestione dei contatti richiedono l'utilizzo di un *pattern* Proxy. Una soluzione più completa avrebbe potuto utilizzare un *pattern* Abstract Factory per gestire le diverse versioni degli algoritmi di *machine learning*. Learner e classifier devono infatti essere utilizzati con versioni omogenee fra loro.

Il diagramma di sequenza segue di conseguenza.

Esercizio 3 (3 punti)

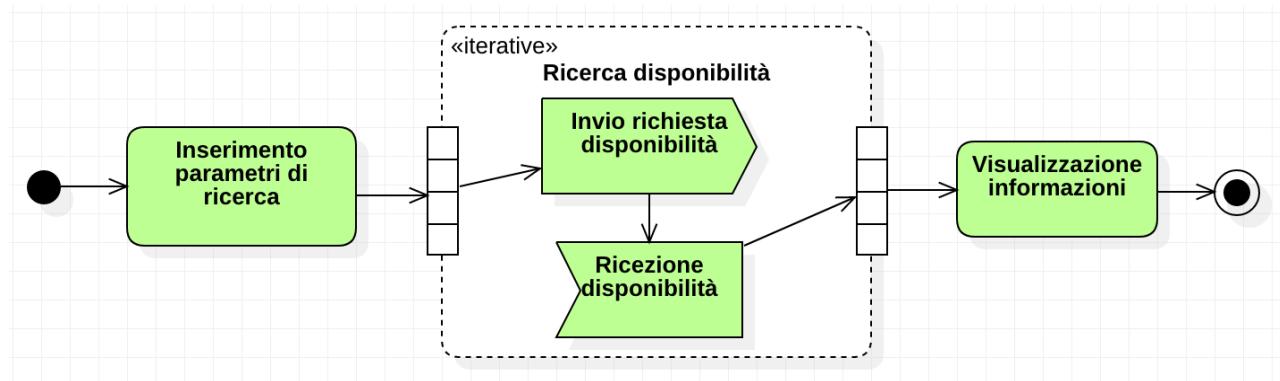
Descrizione

La funzionalità di ricerca all'interno del portale di prenotazione vacanze per famiglie, *Famil.io*, permette di effettuare ricerche relative alla disponibilità di camere, contemporaneamente su più strutture. Dopo aver inserito i parametri di ricerca, il portale invia la corrispondente richiesta a ogni struttura affiliata. Quando tutte le strutture interrogate hanno risposto con la propria disponibilità, il portale raccoglie le informazioni ricevute e le visualizza opportunamente.

Utilizzando un diagramma di attività, si modelli la funzionalità appena descritta.

Soluzione

Una possibile soluzione, che utilizza una regione di espansione e le primitive di invio e ricezione di messaggi, è la seguente.



Ingegneria del Software A.A. 2018/2019

Esame 2019-09-10

Esercizio 1 (6 punti)

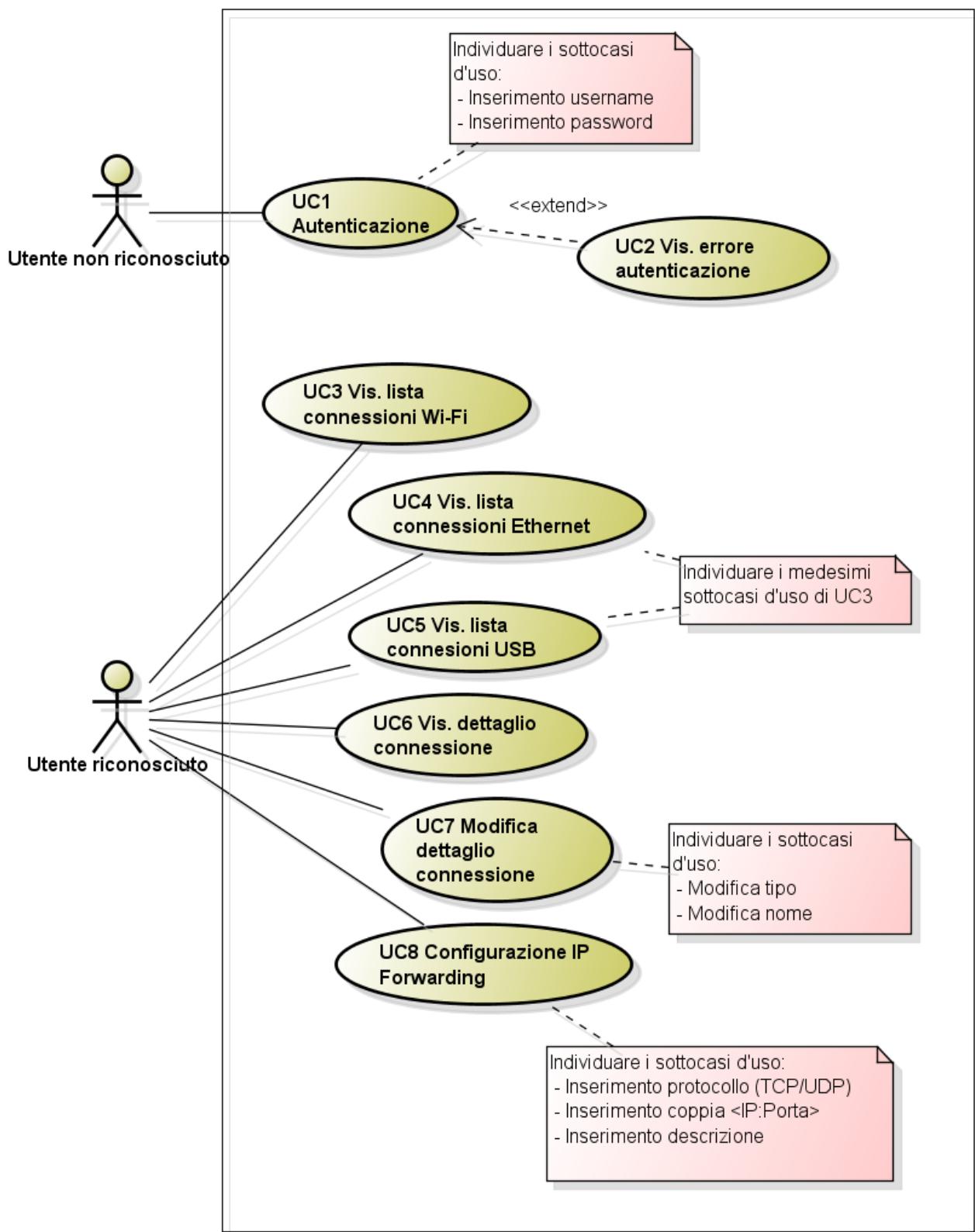
Descrizione

I *router* moderni spesso forniscono una interfaccia web dedicata alla configurazione e disponibile su un indirizzo di rete locale, per esempio 192.168.1.1. Per accedere alla configurazione, è necessario autenticarsi, fornendo *username* e *password*. La *landing page* raggiunta dopo l'autenticazione presenta la situazione delle connessioni di rete attualmente attive per quel *router*, suddivise per tipologia: wi-fi, Ethernet, USB. Ogni lista visualizza un identificativo del dispositivo connesso, e i suoi indirizzi MAC e IPv4. La selezione del singolo dispositivo visualizza informazioni di dettaglio, come lo stato (Connesso/Non connesso), il tipo (Generico/Computer desktop/Computer laptop/ Telefono), e un nome descrittivo. Le ultime due voci possono essere modificate. È inoltre possibile configurare il *port forwarding*, ossia la redirezione di una coppia verso un'altra coppia. Per tale operazione occorre specificare il tipo di protocollo (TCP/UDP), le due coppie, e una breve descrizione testuale.

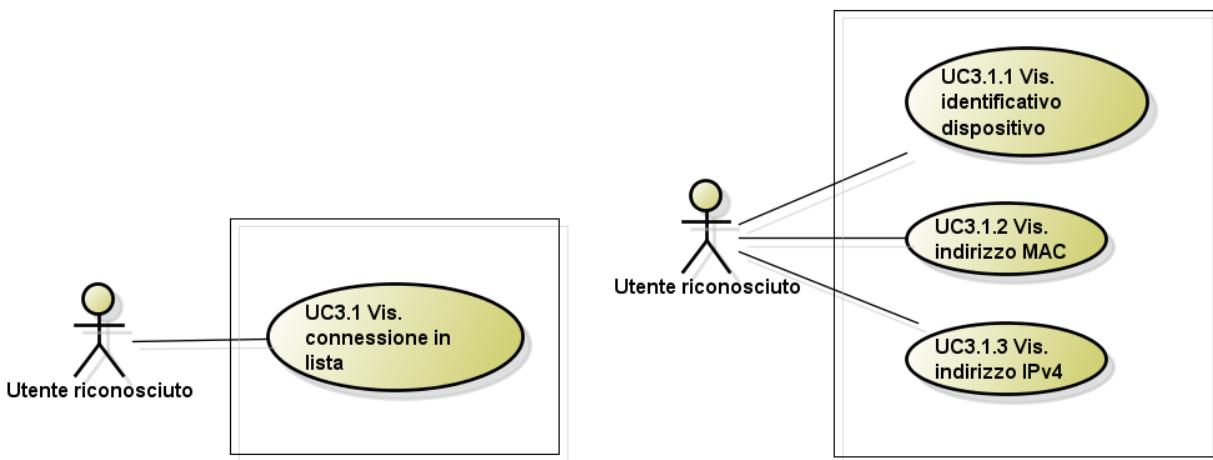
Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari sopra descritti. Non ne è richiesta la descrizione testuale.

Soluzione

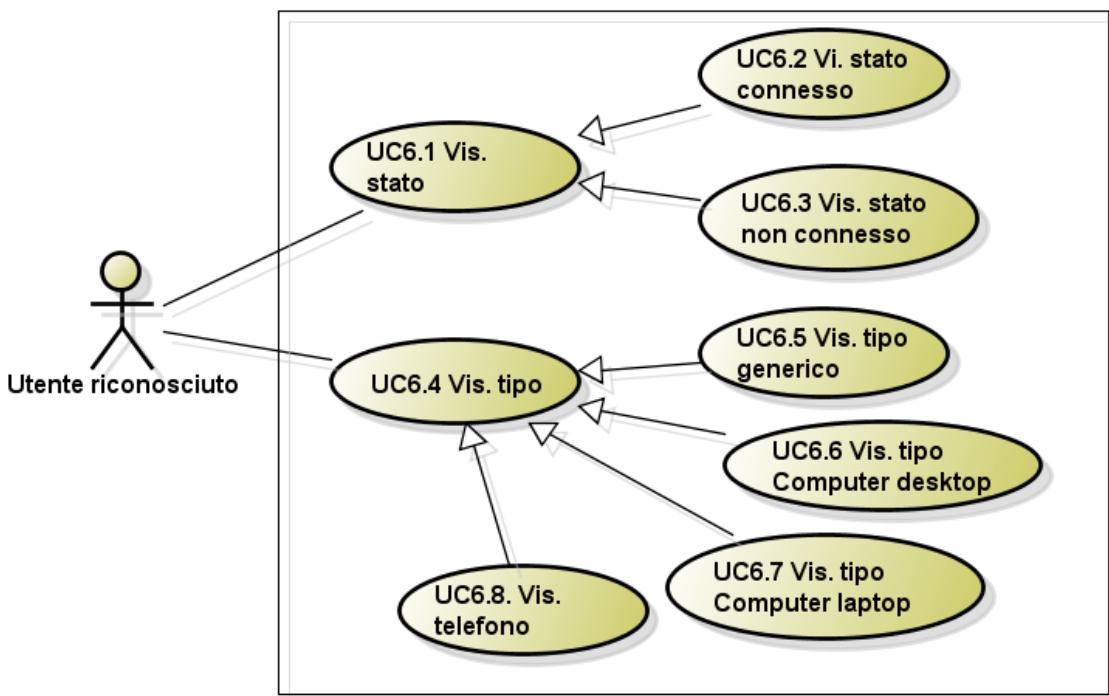
Una possibile soluzione è la seguente.



I sottocasi individuabili sono i seguenti.



E i seguenti.



Esercizio 2 (7 punti)

Descrizione

Una *start-up* innovativa sta sviluppando una stampante che utilizza inchiostri naturali, non inquinanti, per la riproduzione di testo e immagini. Rispetto alla stampa industriale, la particolarità del progetto sta nell'applicazione di differenti algoritmi, flessibilmente adattabili, per la composizione dei colori. Alla base della progettazione software, sta il tipo *Printer* che ha la responsabilità di inviare istruzioni di lavoro alle testine della stampante. Poiché l'algoritmo di selezione dei colori è attualmente ancora in stadio sperimentale, i progettisti dell'azienda hanno necessità di adottare una architettura che ne faciliti l'evoluzione. Gli oggetti da inviare alla stampa sono inizializzati dalle informazioni provenienti da due *listener*, che ascoltano due sonde, poste rispettivamente sulla porta USB e sul connettore wi-fi della stampante. Il tipo di oggetto da stampare viene specificato dall'esterno, e i *listener* semplicemente

acquisiscono e inoltrano tale informazione. La trasformazione degli oggetti in istruzioni di lavoro per le testine avviene utilizzando un componente dedicato. Parte di tali istruzioni sono comuni per i diversi tipi di dato (testo o immagine), parte invece dipende da esso. L'architettura *software* prevede poi un componente che, ricevuto dai *listener* un comando speciale, invia alle testine le istruzioni per effettuare una stampa di prova di un'immagine precaricata nella memoria SSD della stampante.

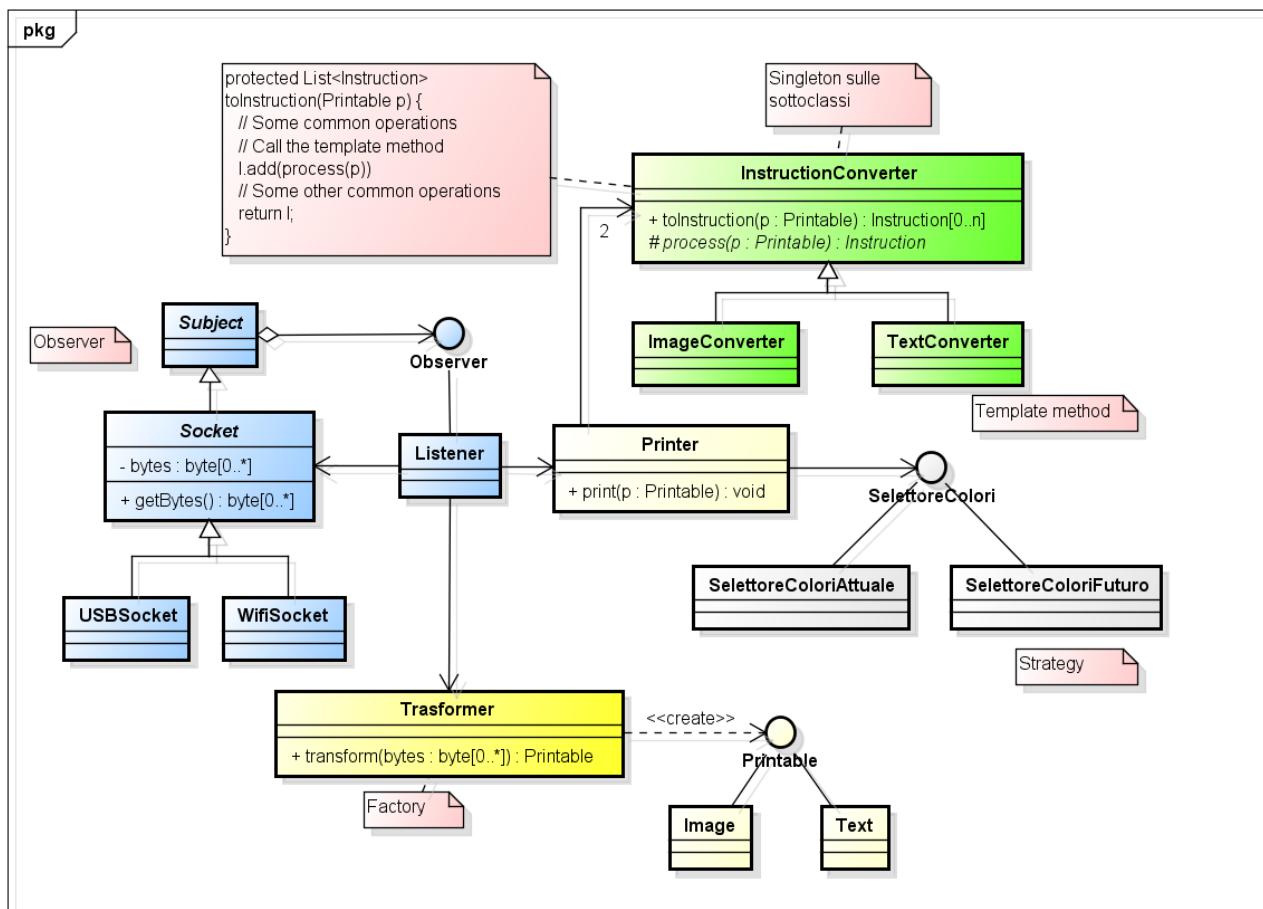
Si modelli tale sistema mediante un diagramma delle classi, comprensivo dei *design pattern* a esso pertinenti. Utilizzando un diagramma di sequenza, si descriva poi la stampa di una immagine.

Soluzione

Una possibile soluzione prevede l'utilizzo dei seguenti *design pattern*:

- Observer
- Strategy
- Template Method
- Factory
- Singleton

Un possibile diagramma delle classi è il seguente.



Il diagramma di sequenza corrispondente è facilmente individuabile.

Esercizio 3 (3 punti)

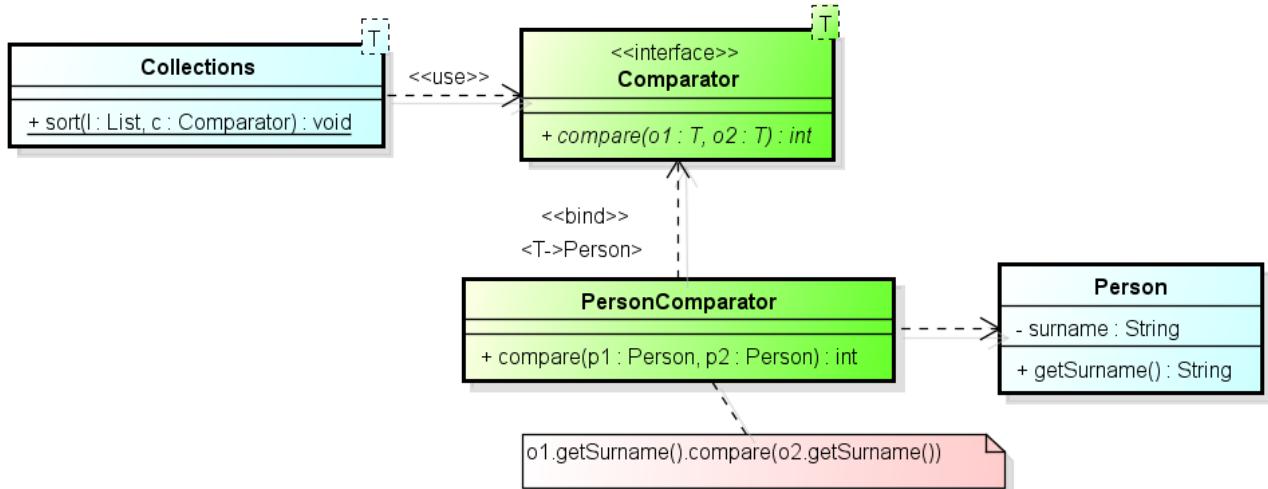
Descrizione

Il *framework* delle *Collection* in Java permette di ordinare una lista in modo arbitrario utilizzando il metodo della classe `Collections, static void sort(List list, Comparator c)`. L'interfaccia `Comparator` contiene un unico metodo `int compare(T o1, T o2)`. Tale *framework* utilizza un noto *design pattern* della GoF.

Si fornisca il diagramma delle classi che lo contestualizza, per una lista di oggetti della classe `Person`, utilizzando un comparatore che ordina per `surname`.

Soluzione

Il *design pattern* utilizzato è lo *Strategy Pattern*. Di seguito si riporta il diagramma delle classi richiesto.



Istruzioni

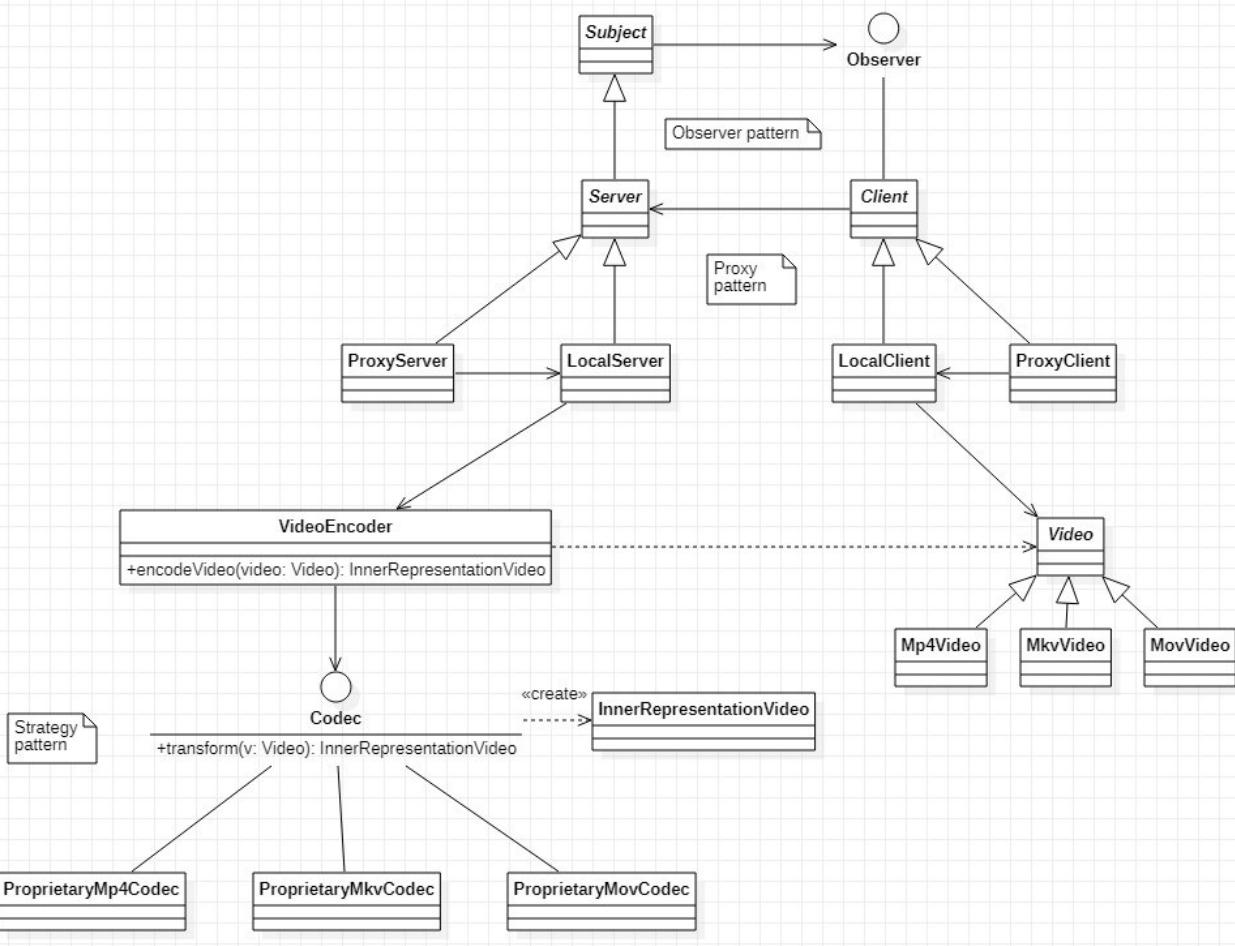
Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome: _____ Nome: _____ Matricola: _____ Anno Progetto Didattico: _____

Domanda 1/3 (punti 5)

Moodle è una piattaforma per l'insegnamento *online*. Tramite essa, è possibile condividere con gli studenti diversi contenuti digitali, in diversi formati. Il *server* accetta video nei formati MP4, MKV o MOV. L'invio del *file* viene effettuato dalla componente *client* verso il *server*, come se esso stesse eseguendo sul medesimo *host*. La comunicazione è però asincrona, ossia al *client* viene ritornato un oggetto che rappresenta il risultato dell'elaborazione che terminerà nel futuro. Il *server* avviserà poi il *client* quando questa "promessa" sarà stata soddisfatta. Il *server* trasforma i vari formati video in ingresso per crearne una propria rappresentazione interna, che utilizza un *codec* proprietario.

Modellare tale sistema utilizzando un diagramma delle classi comprensivo dei *design pattern* a esso pertinenti.

Risposta (Nota sulla soluzione: Cardin commenta che Observer andava fatta sulla Promise)

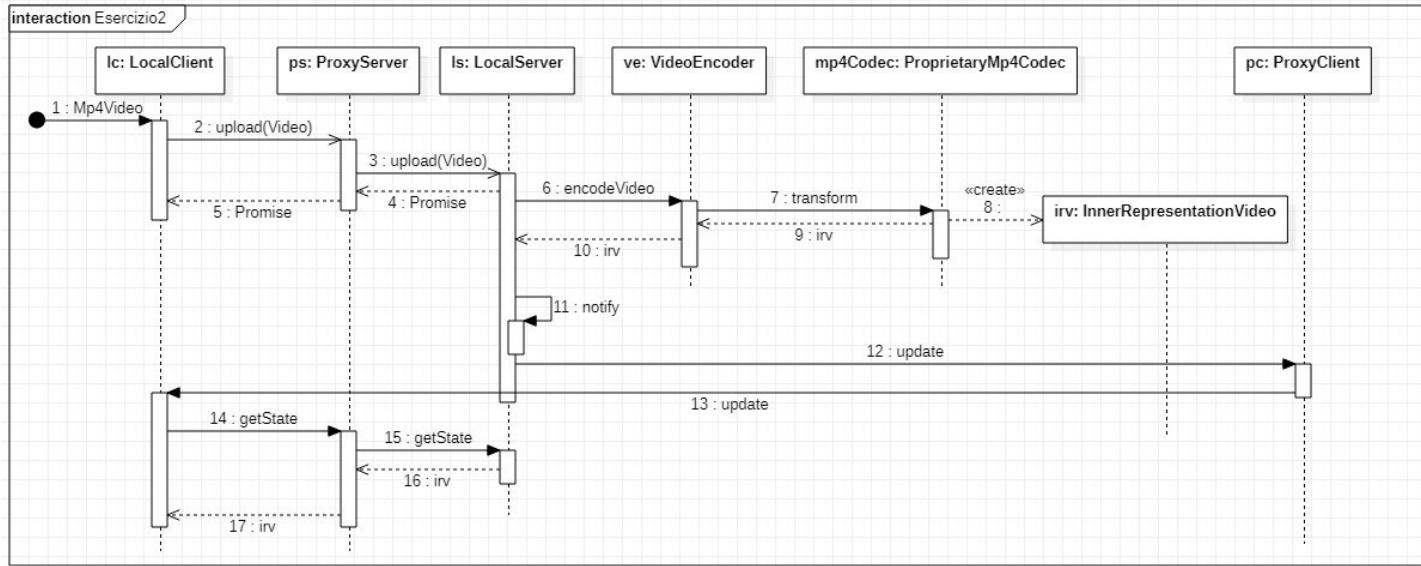
Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome: _____ Nome: _____ Matricola: _____ Anno Progetto Didattico: _____

Domanda 2/3 (punti 3)

Dato il sistema precedentemente descritto, utilizzare un diagramma di sequenza per modellare la collaborazione delle componenti per l'invio di un video in formato MP4 e la successiva soddisfazione della "promessa" da parte del *server*.

Risposta

Istruzioni

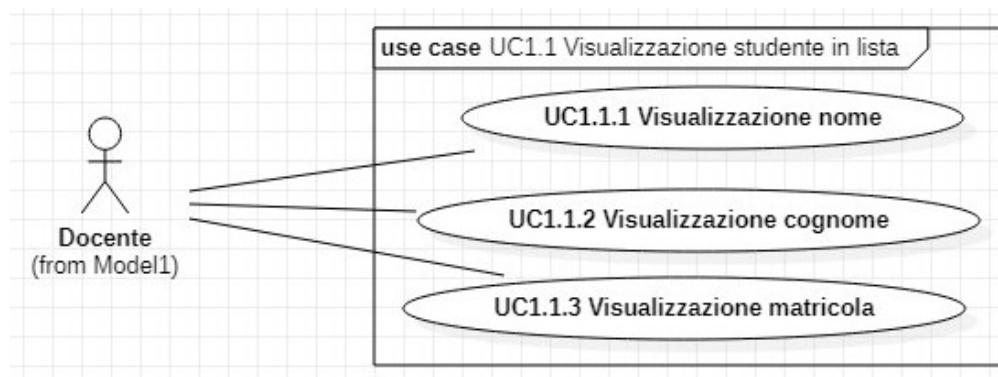
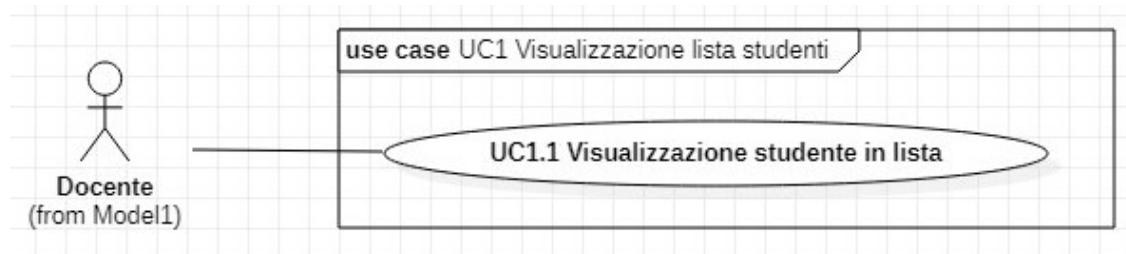
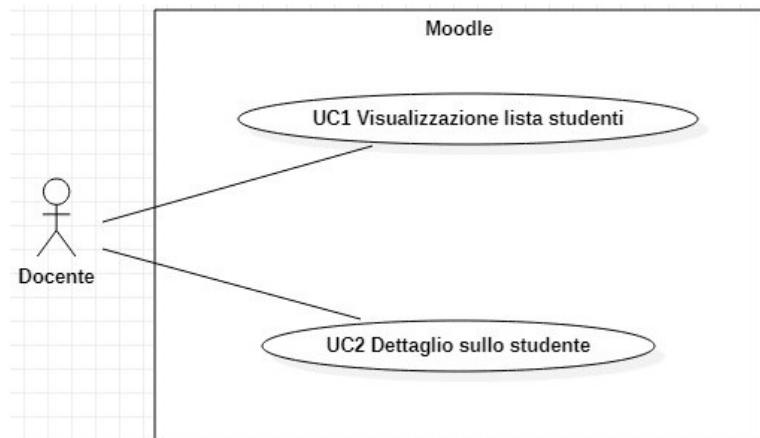
Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome: _____ Nome: _____ Matricola: _____ Anno Progetto Didattico: _____

Domanda 3/3 (punti 2)

Moodle permette ai docenti di visualizzare la lista degli studenti iscritti a un corso. All'interno di tale lista, ogni studente è rappresentato dal proprio nome, cognome e dalla matricola. Selezionando uno studente dalla lista, si possono invece visualizzare le sue informazioni di dettaglio, che riportano anche una foto dello studente e la sua data di nascita.

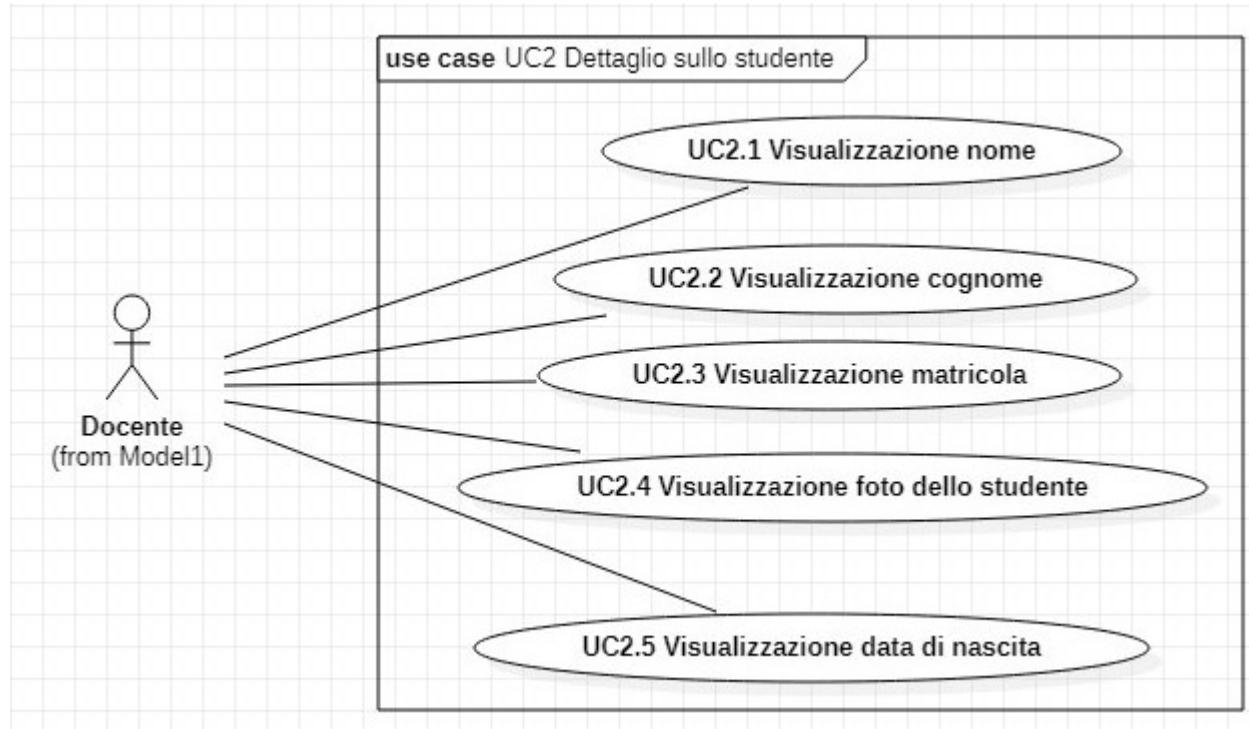
Modellare quanto sopra descritto, utilizzando un diagramma dei casi d'uso. Non è necessaria la descrizione testuale del diagramma.

Risposta

Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome: _____ Nome: _____ Matricola: _____ Anno Progetto Didattico: _____



Ingegneria del Software A.A. 2019/2020

Esame 2020-06-19

Esercizio 1 (6 punti)

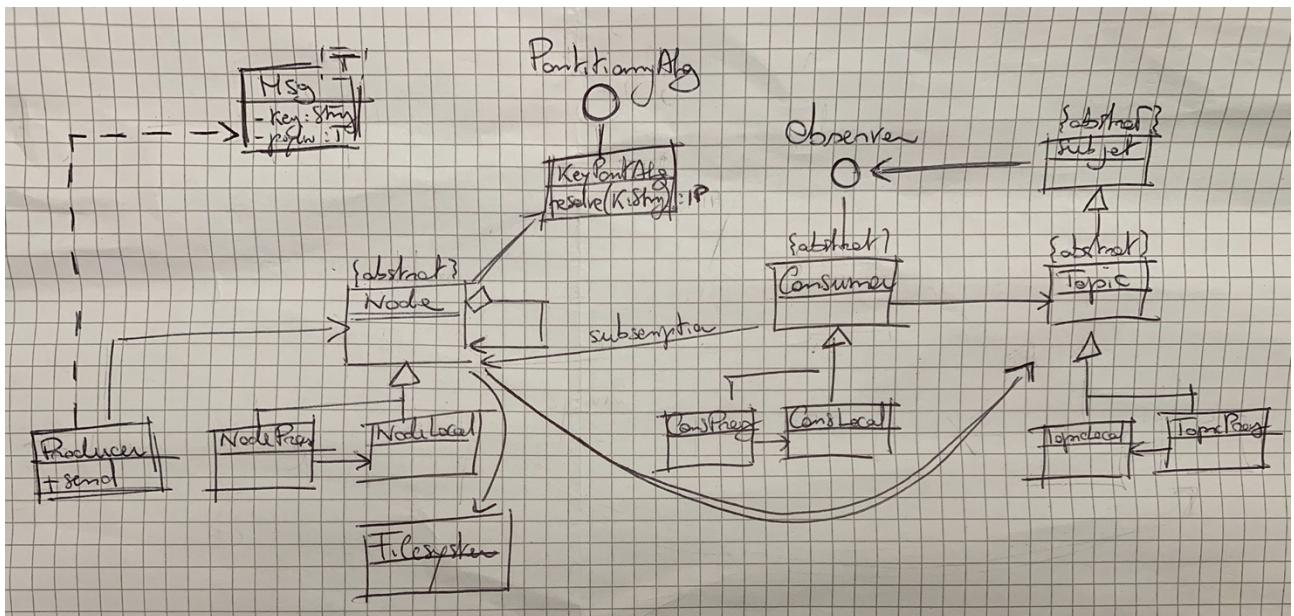
Descrizione

Kafka è un *message broker* che implementa un modello di *logging* distribuito. Ogni log (o messaggio) ha associata una chiave ed un *payload* di tipo T. I messaggi vengono pubblicati su una struttura chiamata *topic*. Un *cluster* è composto da una serie di nodi che si suddividono il lavoro. Ogni nodo gestisce una serie di partizioni per un topic. Le partizioni sono un sottoinsieme delle chiavi possibili per i log. Un nodo è definito *master* ed è responsabile della comunicazione con i *producer* e possiede le tabelle di *routing* per dirottare ogni log alla partizione corretta. Un *producer* invia un log (chiave, valore) per un topic al nodo *master*. La comunicazione avviene via rete. Il nodo *master* utilizza un algoritmo di partizionamento per comprendere a quale altro nodo dirottare il *log*. Questo algoritmo può variare, ma data in input una chiave e un topic restituisce sempre un IP e un numero di partizione. Il nodo *master* invia al nodo deputato il log, che lo salva in un file sequenziale su *filesystem*. Un *consumer*, ossia colui che consuma i messaggi nel *topic*, resta in ascolto da remoto e legge il nuovo log non appena disponibile.

Si modelli tale sistema mediante un diagramma delle classi, comprensivo dei *design pattern* a esso pertinenti.

Soluzione

La soluzione prevede un uso estensivo del pattern Proxy. Inoltre, vengono usati il pattern Observer ed il pattern Strategy.



Esercizio 2 (2 punti)

Descrizione

Dato il sistema precedentemente descritto, si modelli utilizzando un opportuno diagramma di sequenza, la collaborazione delle componenti per l'invio di un nuovo log da parte del nodo *master* ad un altro nodo e la lettura del nuovo messaggio, appena pubblicato, da parte di un ipotetico *consumer*.

Esercizio 3 (2 punti)

Descrizione

Kafka Tool è un'applicazione web che permette di monitorare lo stato di un *cluster* Kafka. Una volta inseriti gli indirizzi IP dei nodi che partecipano al *cluster*, l'applicazione permette di ricercare i log per topic o per consumer. In entrambi i casi, nella lista risultato sono visualizzati per ogni messaggio la chiave, la partizione ed una rappresentazione binaria del *payload*.

Utilizzando un diagramma dei casi d'uso, si modelli quanto descritto. Non è necessaria alcuna descrizione testuale del diagramma.

Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome: Balzan	Nome: Matthew	Matricola: 1193093	Anno Progetto Didattico: 2020
Cognome: Baldisseri	Nome: Michele	Matricola: 1193109	Anno Progetto Didattico: 2020
Cognome: Sassaro	Nome: Giacomo	Matricola: 1187566	Anno Progetto Didattico: 2020

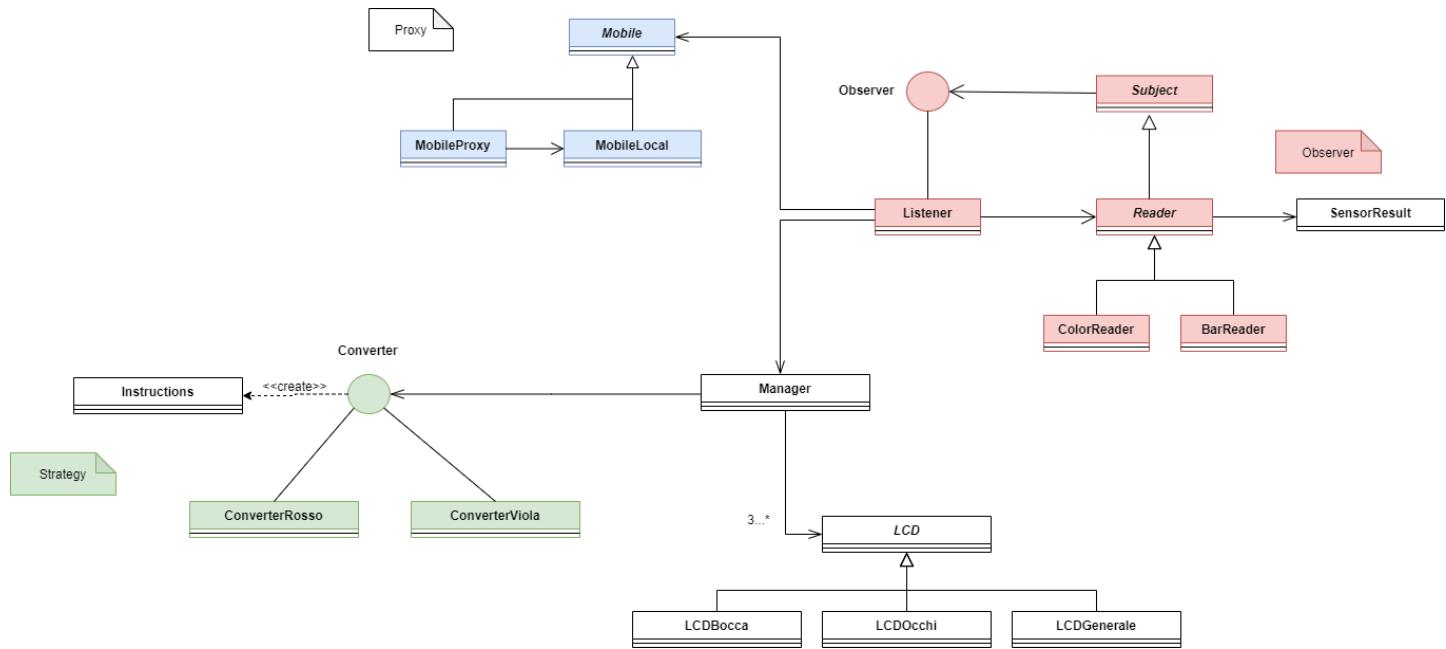
Esito: 9.5/10

Domanda 1/3 (punti 5/30)

La nota casa di mattoncini colorati LEGO ha rilasciato nel 2019 il gioco interattivo LEGO Super Mario in collaborazione con Nintendo. Il gioco è composto da una *mini-figure* di Mario realizzata in parte con i mattoncini LEGO e che dispone di un lettore ottico e di una serie di schermi LCD. Questi schermi visualizzano oltre alle espressioni di occhi e bocca di Mario, anche una serie di informazioni aggiuntive, sulla base di quanto letto dal lettore ottico. In particolare, il lettore è capace di leggere sia colori che codici a barre. Nel caso in cui il lettore individui il colore rosso, l'espressione di Mario (occhi e bocca) sarà di dolore provocato da una bruciatura, mentre il colore viola visualizza uno stato di avvelenamento. L'algoritmo da utilizzare per la visualizzazione richiede sempre l'aggiornamento dello schermo degli occhi, della bocca e quello generale sul torace. Le informazioni raccolte dal lettore, inoltre, vengono inviate via *bluetooth* all'applicazione mobile, che seppur remota, è vista come locale al sistema operativo installato nella *mini-figure*.

Si modelli tale sistema mediante un diagramma delle classi, comprensivo dei *design pattern* a esso pertinenti.

Risposta



Istruzioni

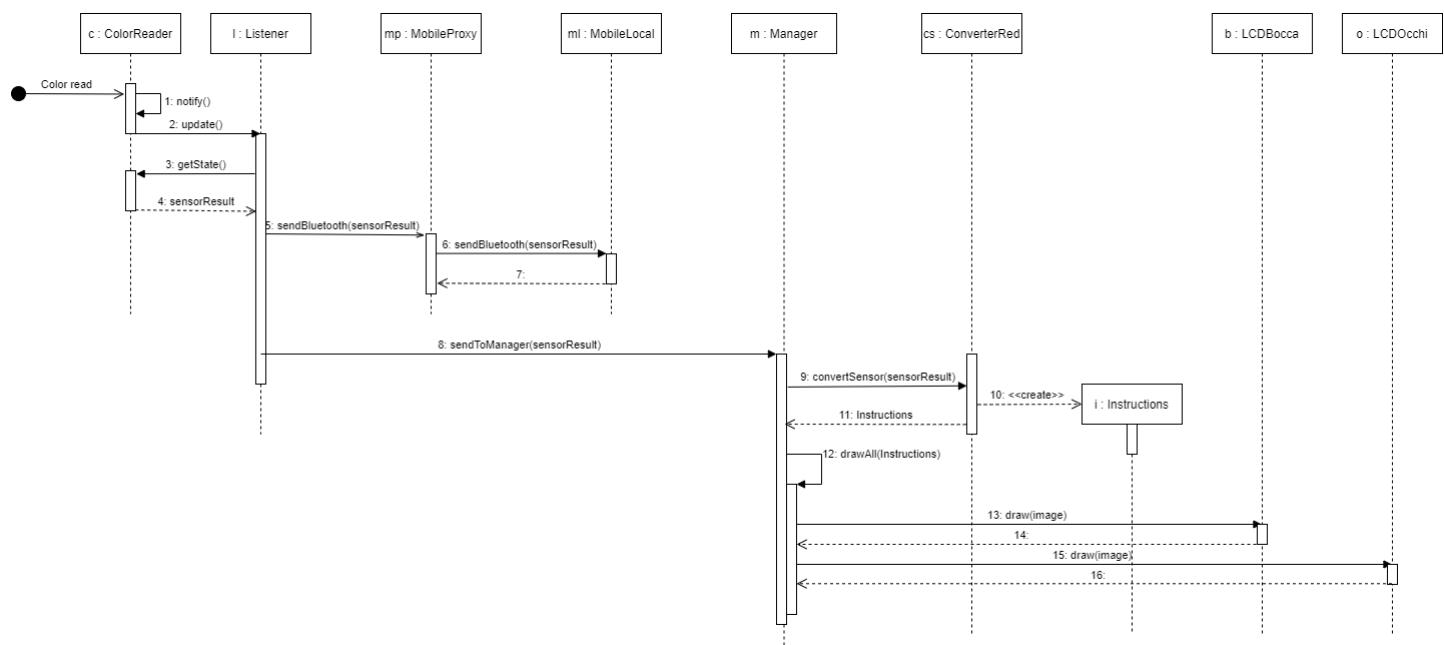
Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome: Balzan	Nome: Matthew	Matricola: 1193093	Anno Progetto Didattico: 2020
Cognome: Baldisseri	Nome: Michele	Matricola: 1193109	Anno Progetto Didattico: 2020
Cognome: Sassaro	Nome: Giacomo	Matricola: 1187566	Anno Progetto Didattico: 2020

Domanda 2/3 (punti 3/30)

Dato il sistema precedentemente descritto, utilizzando un opportuno diagramma di sequenza, si modelli la collaborazione delle componenti coinvolte nella lettura di un colore rosso da parte del lettore ottico.

Risposta



Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

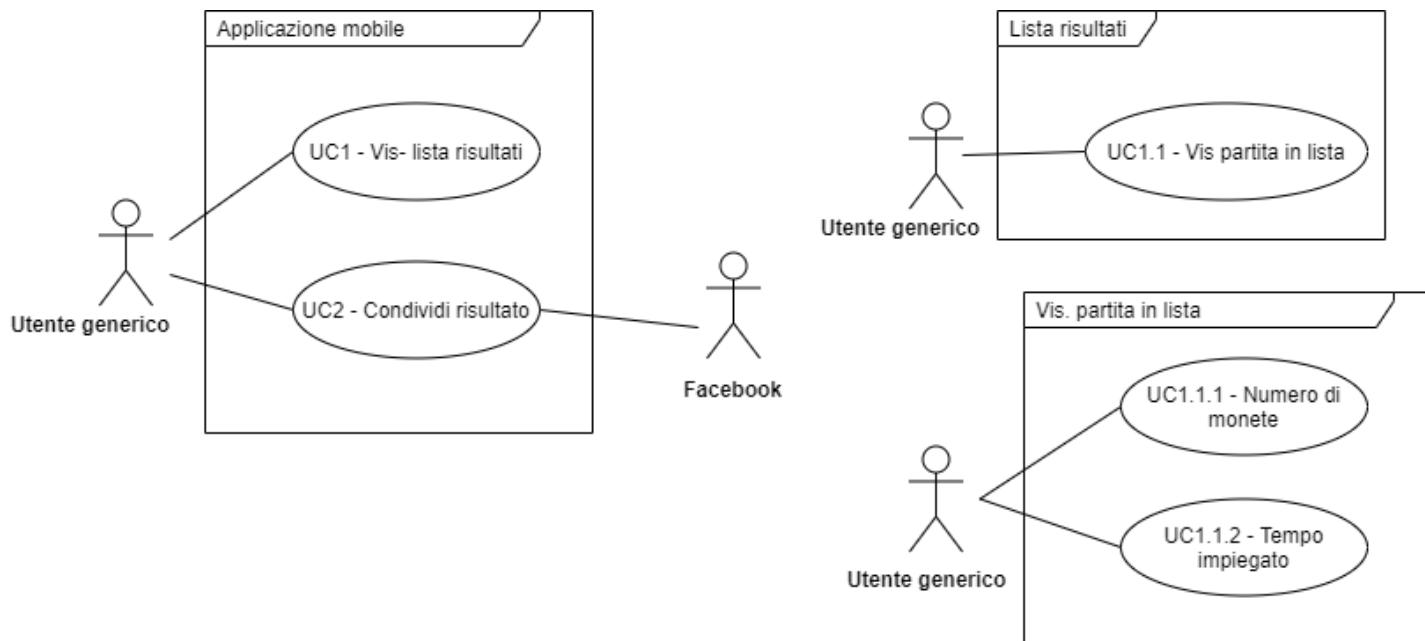
Cognome: Balzan	Nome: Matthew	Matricola: 1193093	Anno Progetto Didattico: 2020
Cognome: Baldisseri	Nome: Michele	Matricola: 1193109	Anno Progetto Didattico: 2020
Cognome: Sassaro	Nome: Giacomo	Matricola: 1187566	Anno Progetto Didattico: 2020

Domanda 3/3 (punti 2/30)

L'applicazione mobile collegata al mondo LEGO Super Mario, permette di visualizzare la lista dei propri risultati collegati ad ogni partita. In particolare, ogni elemento della lista visualizza il numero di monete raccolte e il tempo impiegato per la terminazione del percorso. Inoltre, Ogni elemento della lista visualizza un pulsante tramite il quale è possibile condividere queste informazioni su Facebook.

Utilizzando un diagramma dei casi d'uso, si modellino le esigenze sopra delineate. Non è richiesta alcuna descrizione testuale del diagramma.

Risposta



Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome:	Nome:	Matricola:	Anno Progetto Didattico:
----------	-------	------------	--------------------------

Domanda 1/3 (punti 5/30)

Apple ha recentemente lanciato sul mercato la sua versione dei *tag bluetooth*, chiamata AirTag. Ogni *tag* è un dispositivo fisico capace di connettersi a dispositivi Apple, come iPhone, iPad e MacBook, utilizzando il canale di comunicazione *bluetooth*. A intervalli regolari, ogni *tag* invia un segnale crittato contenente la sua posizione (latitudine e longitudine). Un iPhone che entra nel raggio di azione del *tag*, e che precedentemente era stato abbinato al *tag* stesso, rileva l'informazione con il proprio sensore in ascolto sul canale *bluetooth*. Per facilitarne la gestione, il *tag*, pur non eseguendo fisicamente sul sistema operativo del telefono, è visto come locale ad esso. L'iPhone, quindi, invia a sua volta la posizione del *tag* e il suo identificativo a tutti i dispositivi che si trovino nel suo raggio di azione *bluetooth*. In questo modo, tali telefoni formano una rete estesa, capace di rilevare *tag* anche molto distanti dai telefoni associati. Anche in questo caso, gli iPhone dialogano tra loro come se fossero locali. Infine, un telefono associato a un *tag* può richiedere a quest'ultimo di emettere un suono, utilizzando un opportuno comando, se collegato direttamente alla medesima rete *bluetooth*.

Si modelli tale sistema mediante un diagramma delle classi, comprensivo dei *design pattern* a esso pertinenti.

Risposta

Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome:	Nome:	Matricola:	Anno Progetto Didattico:
----------	-------	------------	--------------------------

Domanda 2/3 (punti 3/30)

Dato il sistema precedentemente descritto, si usi un diagramma di sequenza per modellare la collaborazione delle componenti coinvolte nell'invio da parte di un *tag* della propria posizione a un iPhone, e successivamente, la richiesta da parte di quest'ultimo di emettere un suono.

Risposta

Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome:	Nome:	Matricola:	Anno Progetto Didattico:
----------	-------	------------	--------------------------

Domanda 3/3 (punti 2/30)

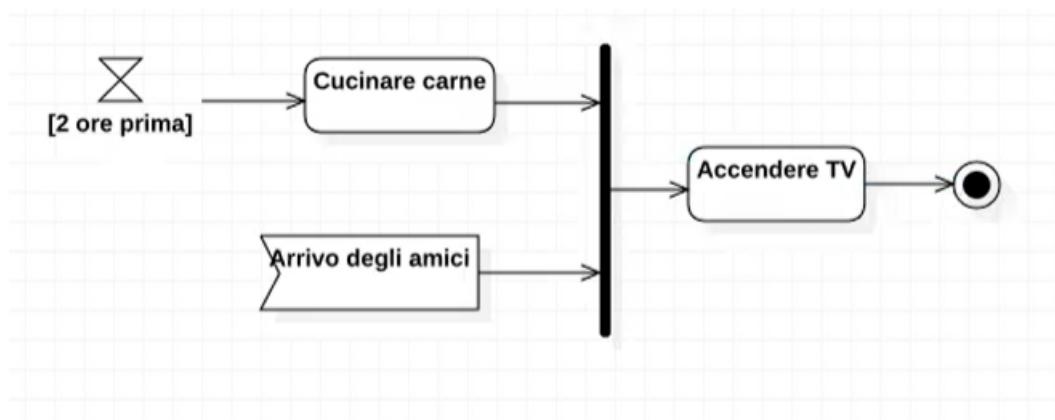
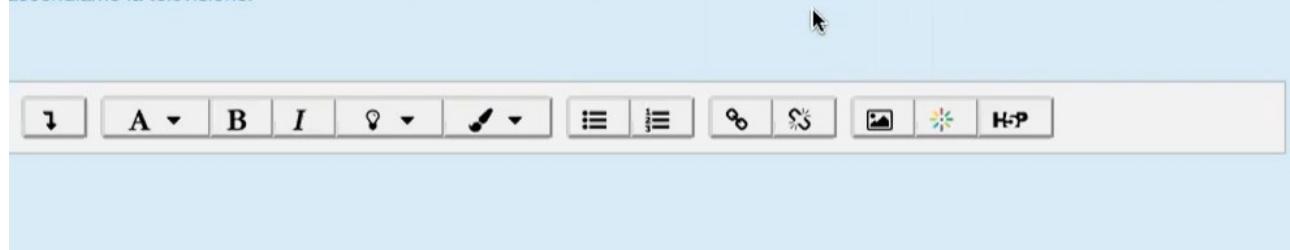
Il sistema operativo iOS fornisce un'applicazione che visualizza, all'interno di una mappa, la posizione degli AirTag a esso associati. In particolare, per ogni *tag* nella mappa, vengono visualizzate latitudine, longitudine e identificativo. Selezionando un *tag* sulla mappa ed entrando nelle informazioni di dettaglio, è possibile richiedere che esso emetta un suono, se questo è collegato direttamente tramite la rete *bluetooth*.

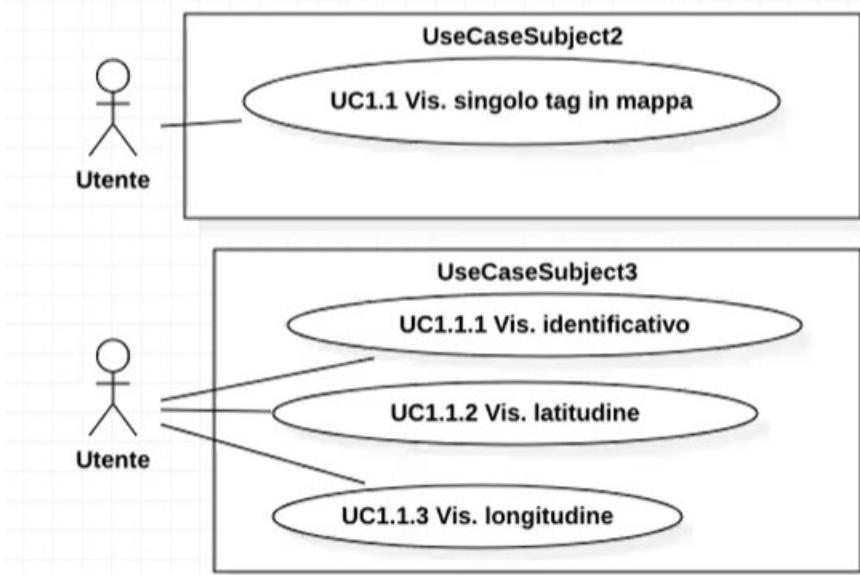
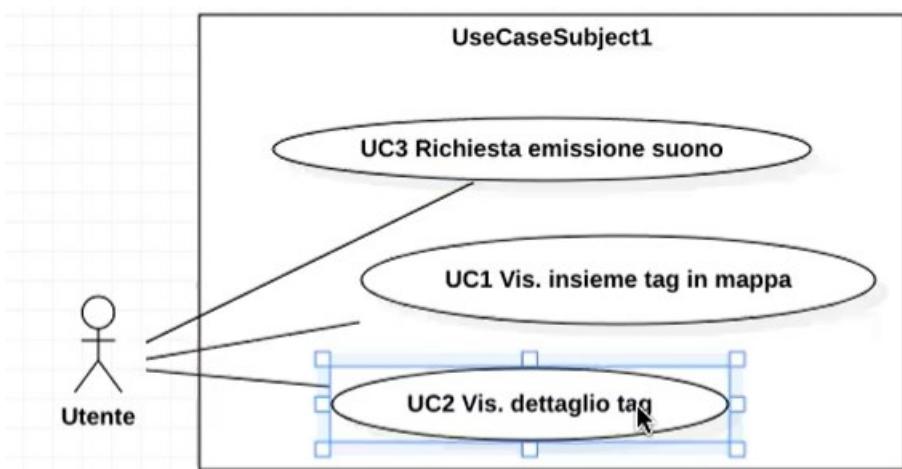
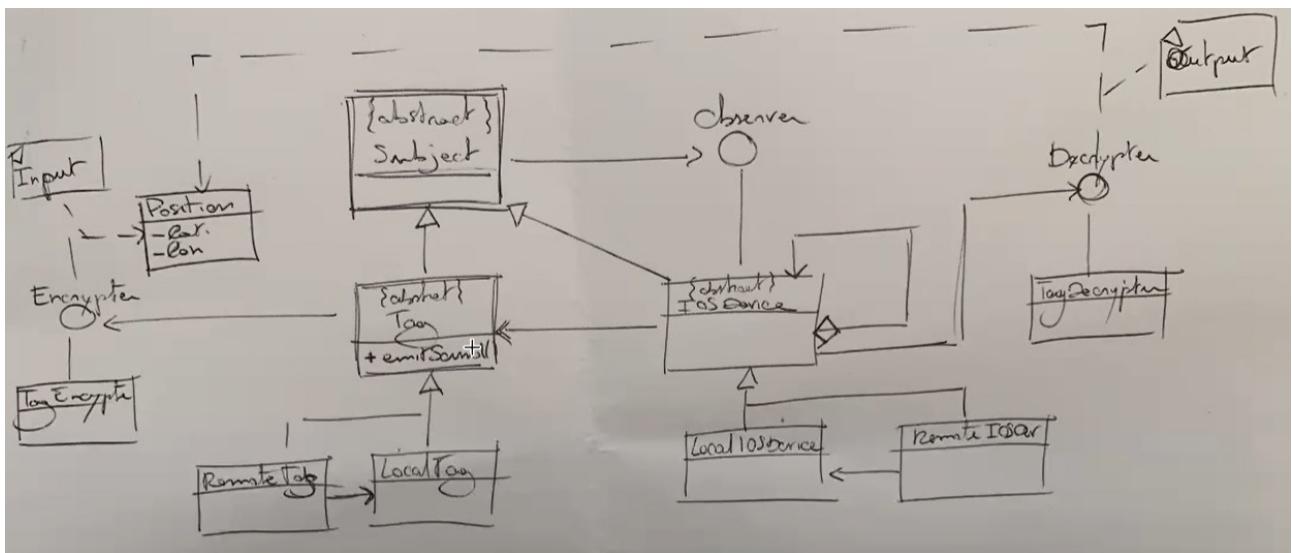
Utilizzando un diagramma dei casi d'uso, si modellino le esigenze sopra delineate. Non è richiesta alcuna descrizione testuale del diagramma.

Risposta

Si modelli utilizzando un diagramma di attività il seguente workflow.

"Due ore prima dell'inizio della partita inizio a cucinare la carne sulla griglia. Quando arrivano anche gli amici e la carne è cotta, allora accendiamo la televisione."





Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome:	Nome:	Matricola:	Anno Progetto Didattico:
----------	-------	------------	--------------------------

Domanda 1/3 (punti 5/30)

Un'azienda di Los Angeles ha sviluppato un sistema capace di interpretare *tweet* pubblicati su *Twitter*, utilizzandone i contenuti per raccomandare l'acquisto o la vendita di pacchetti azionari di una data posizione finanziaria. L'applicazione può essere configurata per restare in ascolto, parallelamente, su specifici profili utenti o *hashtag*. Per ogni nuovo *tweet* pubblicato sui corrispondenti flussi, il sistema lancia un algoritmo di *sentiment analysis* sul testo. Il sistema è progettato in modo che tale algoritmo possa essere facilmente rimpiazzato con una sua versione migliorativa, se e quando disponibile. Una volta ottenuto un risultato dall'algoritmo di analisi, il sistema decide se rafforzare o diminuire specifiche posizioni azionarie. L'interfaccia di compravendita è unica e deve consentire di comprare o vendere azioni usando lo stesso metodo. La piattaforma di compravendita invece è remoto, ma il sistema la usa come se fosse locale.

Si modelli tale sistema mediante un diagramma delle classi, comprensivo dei *design pattern* a esso pertinenti.

Risposta

Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome:	Nome:	Matricola:	Anno Progetto Didattico:
----------	-------	------------	--------------------------

Domanda 2/3 (punti 3/30)

Utilizzando un opportuno diagramma di sequenza, si modelli la collaborazione delle componenti alla pubblicazione di un nuovo *tweet* da parte di un profilo osservato dal sistema precedentemente descritto. Si assuma che il risultato dell'analisi suggerisca l'acquisto di azione Apple AAPL

Risposta

Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome:	Nome:	Matricola:	Anno Progetto Didattico:
----------	-------	------------	--------------------------

Domanda 3/3 (punti 2/30)

Il sistema precedentemente descritto permette all’utente di configurare i profili e gli *hashtag* da seguire e analizzare. A tal fine, il sistema fornisce due distinti modalità di selezione, per tipo di flusso. La lista prodotta da ogni ricerca riporta un pulsante per ogni elemento reperito, che l’utente può utilizzare per suggerire al sistema cosa seguire.

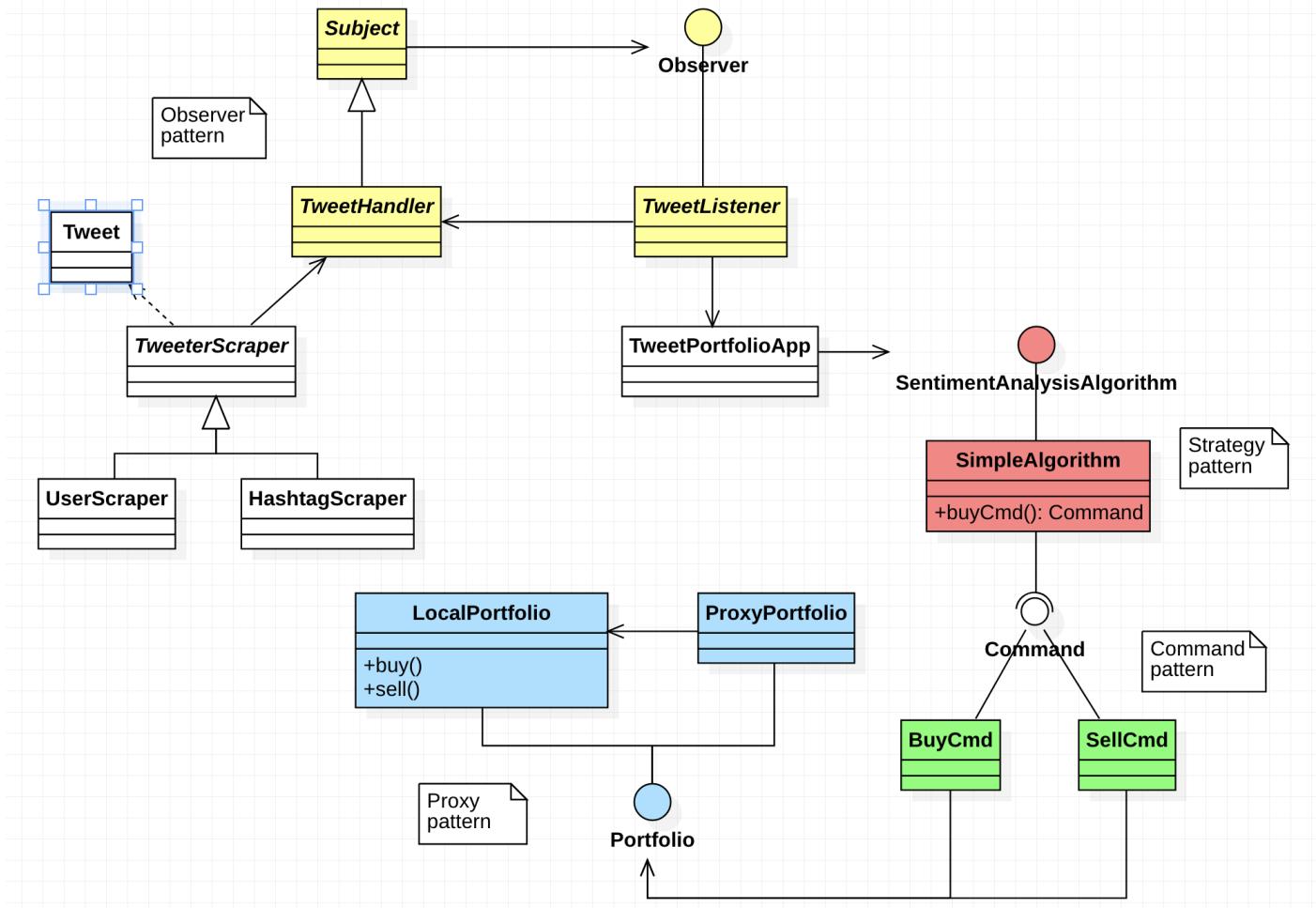
Utilizzando un diagramma dei casi d’uso, si modellino le esigenze sopra delineate. Non è necessaria alcuna descrizione testuale del diagramma.

Risposta

INGEGNERIA DEL SOFTWARE 2021-2022

[Home](#) > [Corsi](#) > AA 2021 - 2022 > Corsi di laurea > INFORMATICA - SC1167 > Ing del Soft > 21 MARZO - 25 MARZO
 > [Soluzione Esercizio 1 dell'Appello del 2020-07-20](#)

Soluzione Esercizio 1 dell'Appello del 2020-07-20



A differenza di quanto visto a lezione, la soluzione presentata differisce per alcuni punti:

1. Rileggendo con attenzione il testo, si può comprendere che il sistema remoto è il sistema di compravendita.
2. Per non utilizzare l'ereditarietà, è possibile utilizzare un approccio alternativo che utilizza la composizione per lo *scraping* dei tweet derivanti da uno *user* o un *hashtag*.
3. Come promesso, è stato aggiunto il *pattern Command* mancante.

◀ [Registrazione della lezione E01 \(anno 2020/2021\)](#)

Vai a...

[Quiz \(Teoria\) ►](#)

DOCUMENTAZIONE

Moodle

Kaltura



Riepilogo della conservazione dei dati

Ottieni l'app mobile

Politiche

Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome:	Nome:	Matricola:	Anno Progetto Didattico:
----------	-------	------------	--------------------------

Domanda 1/3 (punti 5/30)

Simple Storage Service (S3) è il servizio di Amazon AWS che fornisce uno spazio di archiviazione virtualmente illimitato su *cloud*. In S3, ogni *file*, o aggregato di dati, è contenuto in un *bucket*. In tale *bucket* ogni *file* è contraddistinto da un prefisso (*prefix*), ossia un percorso simile a quello di un *file system*, e da un nome. Un *file* viene inviato ad S3 utilizzando una richiesta HTTP utilizzando il verbo PUT. AWS pone un limite al numero massimo di richieste effettuabili da un singolo *client*, fissato a 3500 richieste di inserimento al secondo per singolo prefisso. Ad esempio, il prefisso /2022/12/photo/ può ricevere non più di 3500 req/s di inserimento di oggetti. Nel caso in cui un *client* ecceda tale limite, AWS risponde alle richieste in eccesso con lo stato HTTP 503 (*throttling*) fino a quando il *client* non diminuisce il *rate* delle richieste riportandolo entro il limite fissato. L'algoritmo di *throttling* in S3 è molto complesso e viene migliorato di versione in versione. I *client* di S3 sono di molteplici tipi. Il più semplice è la AWS CLI, cioè una semplice riga di comando. La CLI permette di inserire in S3 un *file* locale alla *workstation* nella quale essa è attiva. Il codice della CLI resta in attesa del comando di inserimento e interagisce con il *server* di S3 come se fosse locale a esso. La CLI è stata programmata dagli sviluppatori di AWS utilizzando il *pattern* MVC, per permettere chiara divisione dei compiti delle sue varie componenti.

Si modelli tale sistema mediante un diagramma delle classi, comprensivo dei *design pattern* a esso pertinenti.

Risposta

Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome:	Nome:	Matricola:	Anno Progetto Didattico:
----------	-------	------------	--------------------------

Domanda 2/3 (punti 3/30)

Dato il sistema descritto nel quesito precedente, usate un diagramma di sequenza per modellate la collaborazione delle componenti coinvolte nel trattamento di una richiesta di invio *file* a S3 effettuata da AWS CLI.

Risposta

Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

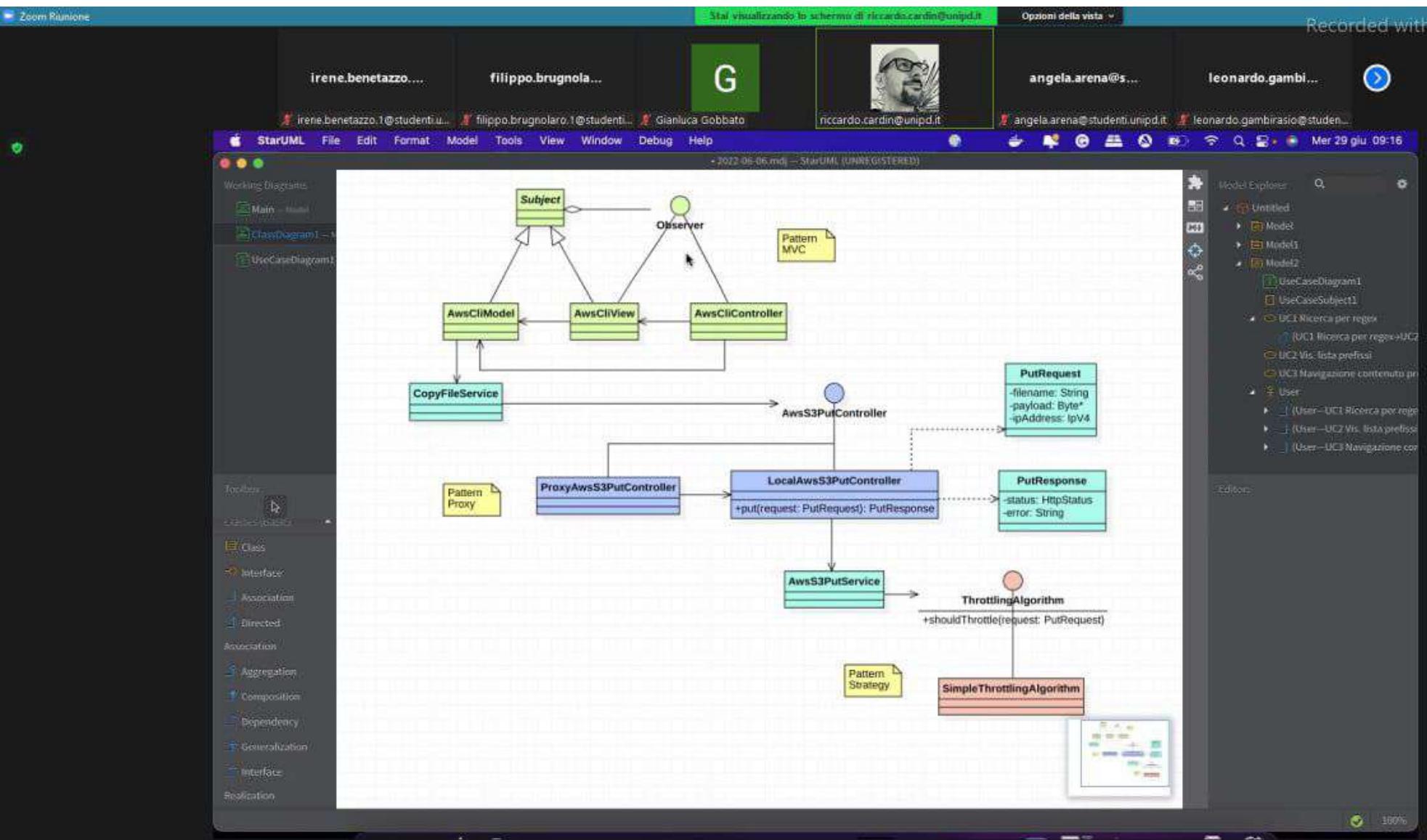
Cognome:	Nome:	Matricola:	Anno Progetto Didattico:
----------	-------	------------	--------------------------

Domanda 3/3 (punti 2/30)

AWS S3 dispone anche di una interfaccia *web*, che permette di navigare attraverso il *file system* interno a S3. Con tale interfaccia è possibile ricercare un prefisso tramite una espressione regolare. Il risultato della ricerca è una lista di prefissi che soddisfano tale espressione. Ciascun prefisso nella lista visualizza il nome, la data di ultimo accesso e la data di creazione. Selezionando il prefisso, è poi possibile navigarne il contenuto.

Modellare le esigenze sopra delineate utilizzando un diagramma dei casi d'uso. Non è richiesta descrizione testuale del diagramma.

Risposta



irene.benetazzo.... filippo.brugnola... G riccardo.cardin@unipd.it pietro.macri@st... ruthgenevieve.bousapnamene...

StarUML File Edit Format Model Tools View Window Debug Help

Working Diagrams Main - model ClassDiagram1 UseCaseDiagram1 SequenceDiagram

Interacted SequenceDiagram

```
sequenceDiagram
    actor1->>Lifeline1: Message1
    activate Lifeline1
    Lifeline1->>Lifeline2: Message2
    activate Lifeline2
    Lifeline2->>Lifeline3: Message3
    activate Lifeline3
    Lifeline3->>Lifeline4: Message4
    activate Lifeline4
    Lifeline4->>Lifeline5: Message5
    activate Lifeline5
    Lifeline5->>Lifeline6: Message6
    activate Lifeline6
    Lifeline6->>Lifeline7: Message7
    activate Lifeline7
    Lifeline7->>Lifeline8: Message8
    activate Lifeline8
    Lifeline8->>Lifeline9: Message9
    activate Lifeline9
    Lifeline9->>Lifeline10: Message10
    activate Lifeline10
    Lifeline10->>Actor1: Message11
    deactivate Lifeline10
    deactivate Lifeline9
    deactivate Lifeline8
    deactivate Lifeline7
    deactivate Lifeline6
    deactivate Lifeline5
    deactivate Lifeline4
    deactivate Lifeline3
    deactivate Lifeline2
    deactivate Lifeline1
```

Model Explorer

- Message1 (Lifetime1 < Lifetime2)
- Message2 (Lifetime2 < Lifetime3)
- Message3 (Lifetime3 < Lifetime4)
- Message4 (Lifetime4 < Lifetime5)
- Message5 (Lifetime5 < Lifetime6)
- Message6 (Lifetime6 < Lifetime7)
- Message7 (Lifetime7 < Lifetime8)
- Message8 (Lifetime8 < Lifetime9)
- Message9 (Lifetime9 < Lifetime10)
- Message10 (Lifetime10 < Lifetime1)

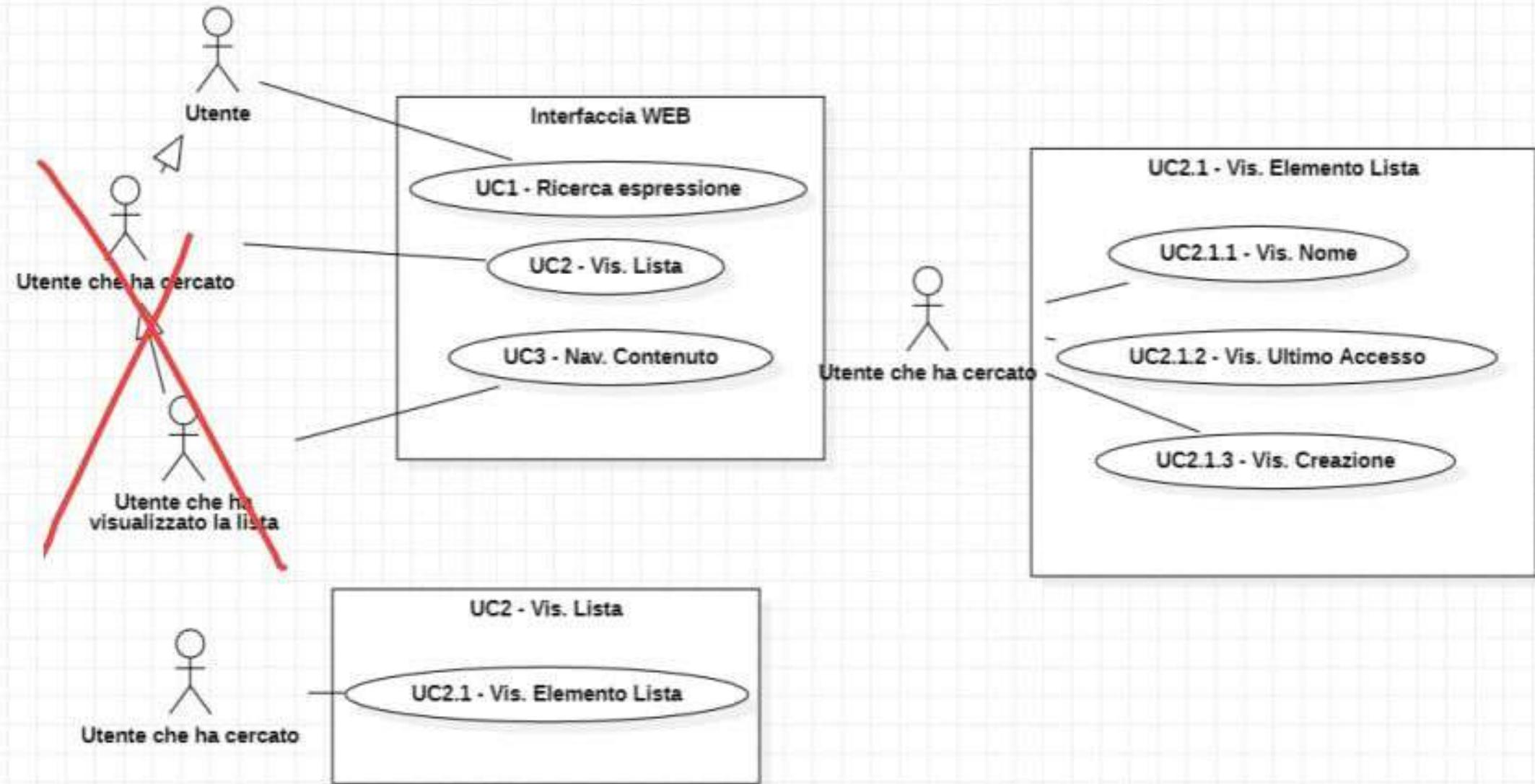
Toolbox

- Found Message
- Last Message
- Endpoint
- Gate
- State Invariant
- Continuation
- Combined
- Fragment
- Interaction Use
- Frame
- Annotations

Collaboration1 Interaction1 SequenceDiagram1 UML2.5.0.20220629-0936

name: SequenceDiagram
defaultDiagram: 50%

09:36 29/06/2022



Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome:	Nome:	Matricola:	Anno Progetto Didattico:
----------	-------	------------	--------------------------

Angeloni Alberto 1231122 22/23

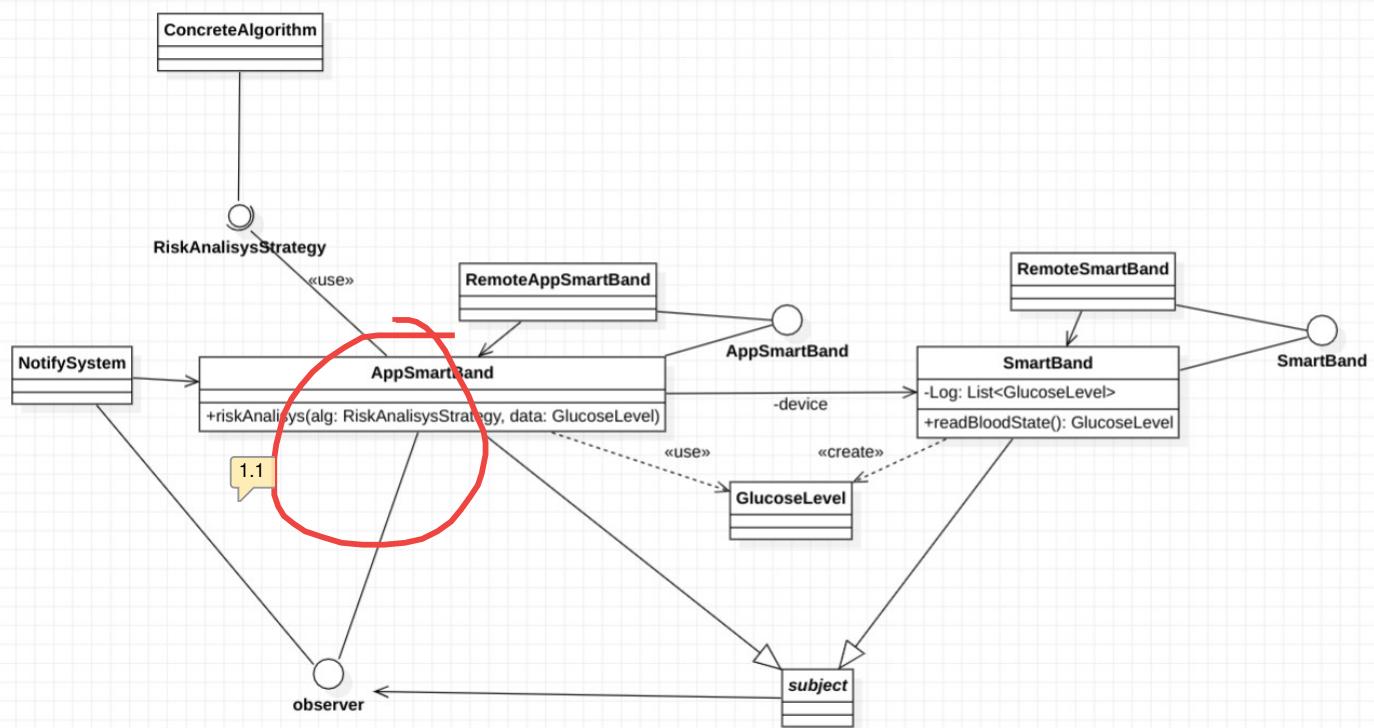
Angeli Jacopo 1232583 22/23

Domanda 1/2 (punti 6/30)

Gli *smartband* sono dispositivi indossabili che monitorano lo stato di salute e le condizioni fisiche di una persona. Una *startup* italiana sta sviluppando un nuovo dispositivo che monitora il livello di glucosio nel sangue. Il dispositivo effettua una misurazione ogni 5 minuti, persistendo le informazioni su una memoria volatile locale. Ogni 30 minuti, il dispositivo si sincronizza con l'applicazione gemella installata sullo *smartphone*, la quale resta permanentemente in ascolto delle informazioni provenienti dallo *smartband*. La comunicazione tra *app* e dispositivo avviene tramite rete *Bluetooth*, ma le due parti interagiscono come se fossero locali l'una all'altra. Ricevuto l'aggiornamento, l'*app* ne utilizza i dati per calcolare il livello di rischio del paziente, utilizzando un algoritmo proprietario, ancora oggetto di sviluppo, e pertanto frequentemente soggetto ad aggiornamenti. Qualora il livello di rischio rilevato fosse elevato, l'*app* avvisa l'utente visualizzando a schermo un messaggio di allarme.

Si modelli tale sistema mediante un diagramma delle classi, comprensivo dei *design pattern* a esso pertinenti.

Risposta



Esame scritto – III prova scritta - parte Compito (**Pratica**) – 21 giugno 2023

Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome: Nome: Matricola: Anno Progetto Didattico:

Angeloni Alberto 1231122 22/23

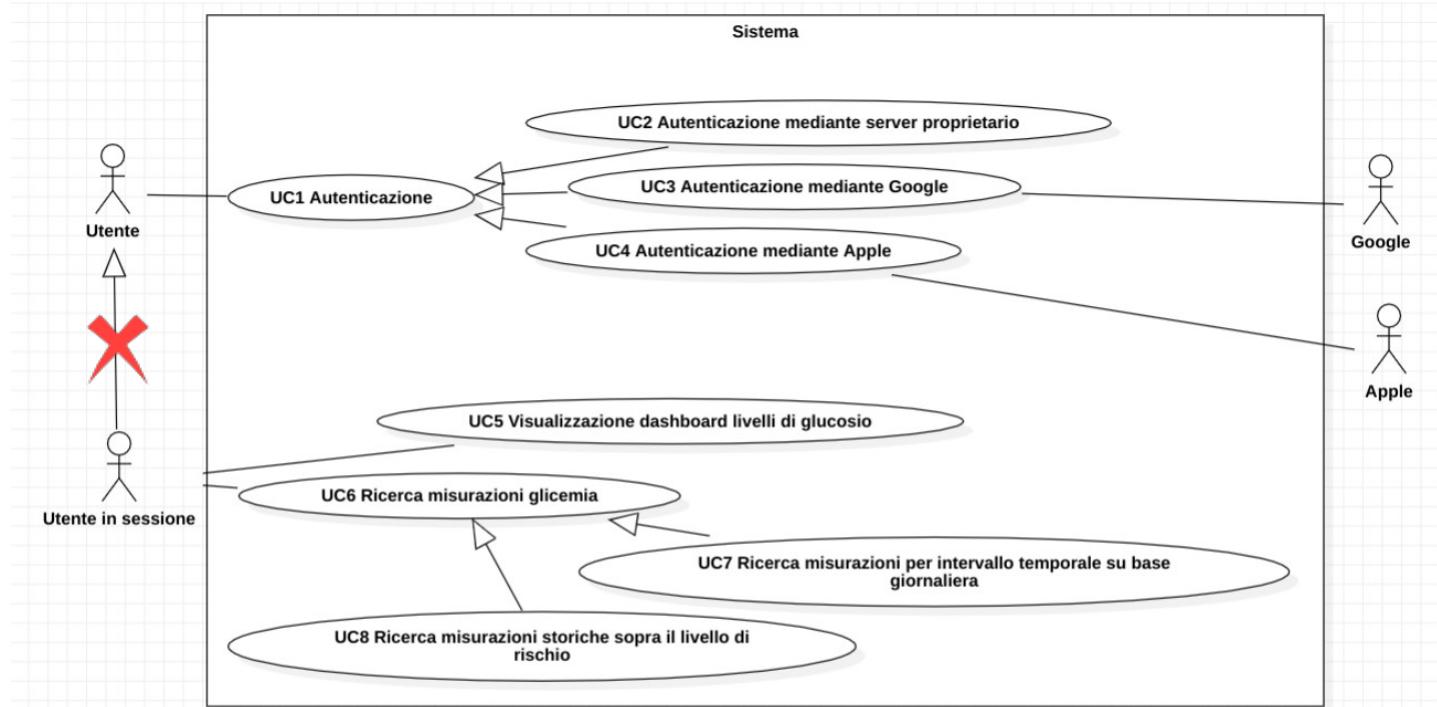
Angeli Jacopo 1232583 22/23

Domanda 2/2 (punti 4/30)

L'app di lato *smartphone* che monitora i livelli di glucosio, richiede autenticazione per consentire all'utente di interagire con essa. Tale autenticazione può avvenire utilizzando un *server proprietario* o i servizi offerti da Google e Apple. Una volta autenticati, gli utenti visualizzano una *dashboard* contenente l'andamento dei livelli di glucosio nella giornata corrente. Il grafico giornaliero riporta il livello misurato e quello atteso. L'utente può scegliere quali dati visualizzare, ricercandoli per intervallo temporale su base giornaliera, oppure restringere la visualizzazione ai soli rilievi storici che abbiano superato il livello di rischio, e quindi abbiano generato un avviso.

Modellare le esigenze sopra delineate utilizzando un diagramma dei casi d'uso. Non è richiesta descrizione testuale del diagramma.

Risposta



**Università di Padova – Informatica – Ingegneria del Software
pag. 3 / 3**

Esame scritto – III prova scritta - parte Compito (**Pratica**) – **21 giugno 2023**

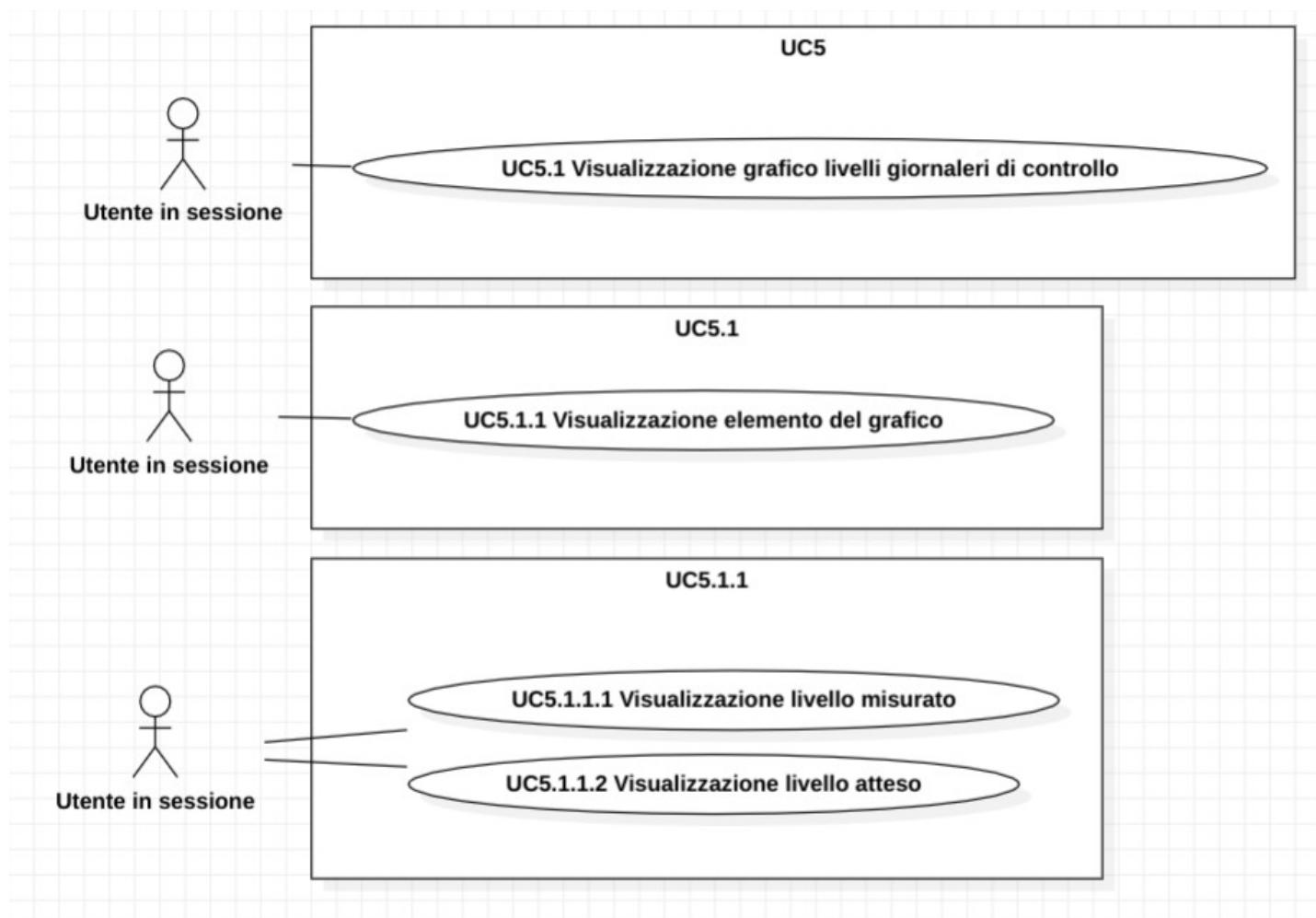
Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome: Nome: Matricola: Anno Progetto Didattico:

Angeloni Alberto 1231122 22/23

Angeli Jacopo 1232583 22/23



Indice dei commenti

1.1 Sono concentrate molte responsabilità in questa classe

Istruzioni

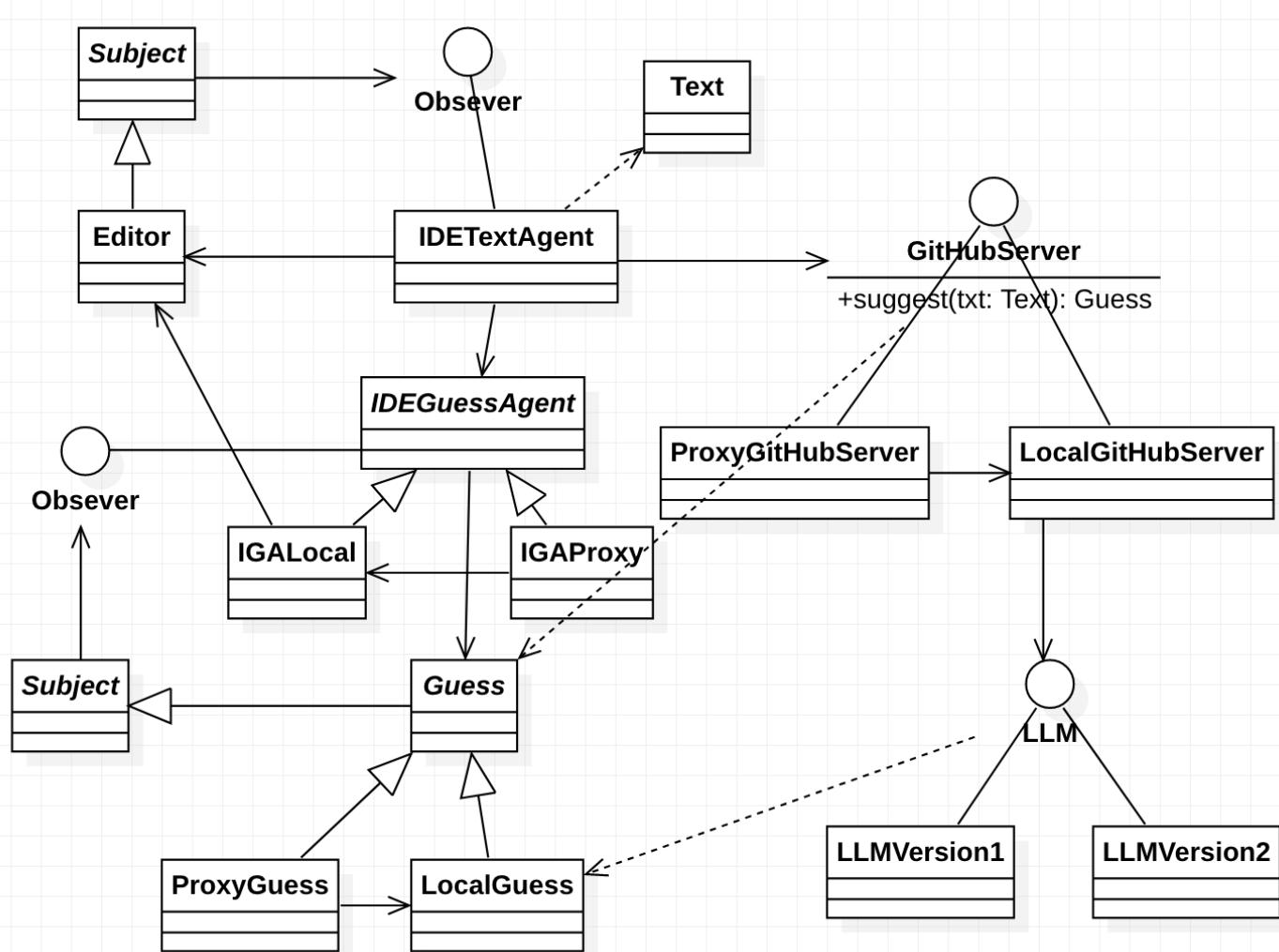
Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome: _____ Nome: _____ Matricola: _____ Anno Progetto Didattico: _____

Domanda 1/2 (punti 6/30)

GitHub Copilot è una Intelligenza Artificiale fortemente mirata al supporto dello sviluppo *software*. Un agente installato sull’IDE dello sviluppatore resta in ascolto dei caratteri digitati nell’*editor*. Man mano che l’utente digita testo sul proprio dispositivo, esso (testo) viene inviato a uno dei *server* GitHub dedicati della piattaforma. Questo trasferimento avviene come se il *server* destinazione fosse locale all’agente, pur essendo remoto. Quando il *server* ha ricevuto i contenuti attesi, li tratta utilizzando un modello di LLM (*Large Language Model*) che fornisce il prossimo suggerimento da fornire allo sviluppatore. L’agente resta in ascolto del prossimo suggerimento disponibile e lo visualizza nell’*editor*. Il suggerimento viene notificato all’agente attraverso un canale dedicato e aperto all’avvio della sessione. Anche in questo caso, la comunicazione avviene simulando uno scambio locale, pur essendo remota. Il modello di LLM utilizzato in questo sistema è in continua evoluzione: diverse versioni ne devono essere rilasciate periodicamente, senza però che ciò comporti una continua re-ingegnerizzazione dell’applicazione.

Si modelli tale sistema mediante un diagramma delle classi, comprensivo dei *design pattern* a esso pertinenti.

Risposta

Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

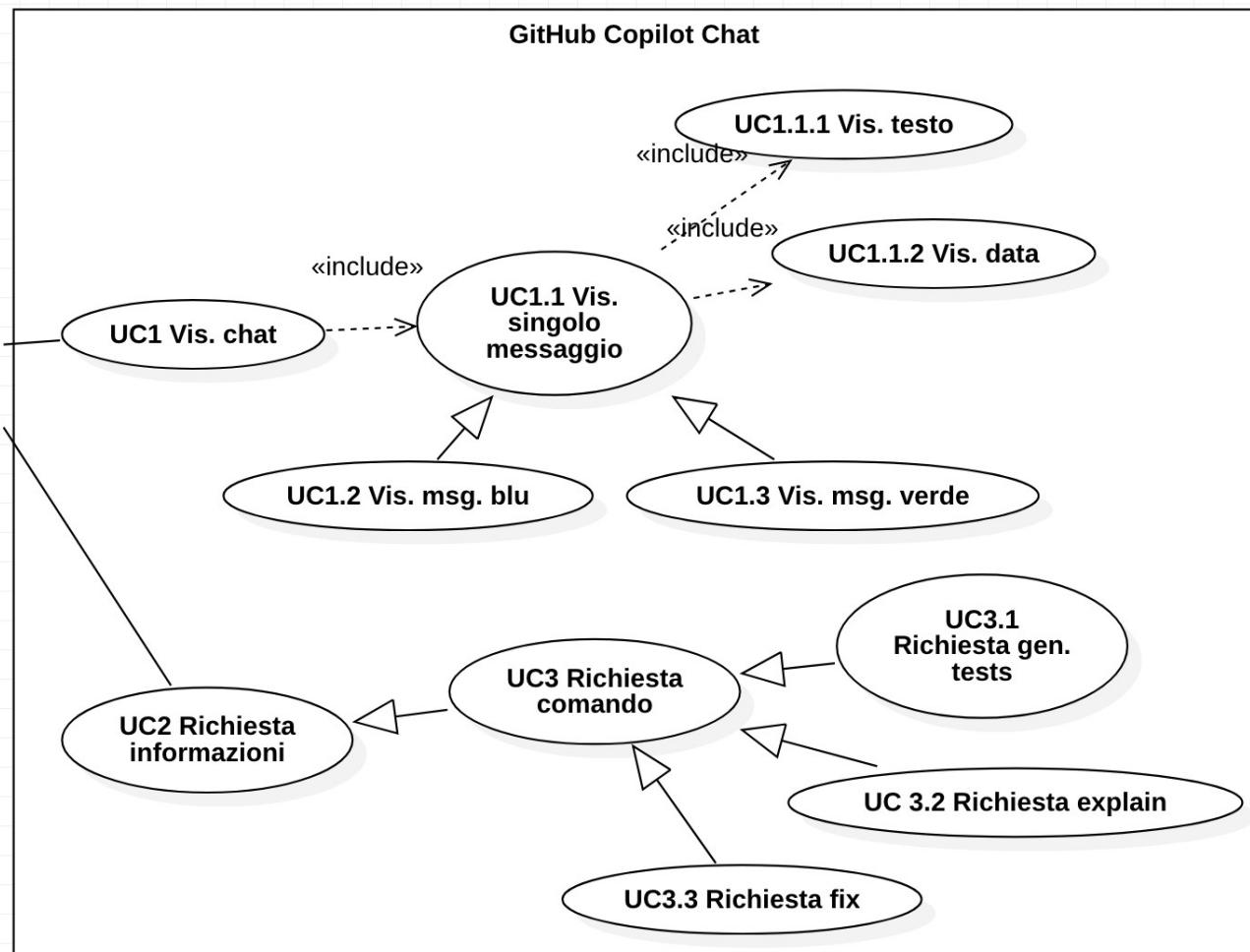
Cognome: _____ Nome: _____ Matricola: _____ Anno Progetto Didattico: _____

Domanda 2/2 (punti 4/30)

GitHub ha da poco rilasciato in versione *alfa* un assistente virtuale simile a ChatGPT, ossia consultabile attraverso una *chat*. Nella *chat* sono visualizzati tutti i messaggi scritti fin a quel momento. Del messaggio viene riportato il testo e la data in cui esso è stato inviato. I messaggi si distinguono in: messaggi dell’utente, di colore *blu*; e messaggi scritti direttamente da *GitHub Copilot Chat*, di colore verde. Lo sviluppatore può richiedere a *Copilot* qualsiasi informazione, inviando un nuovo messaggio nella *chat*. Lo sviluppatore può anche accedere a una serie di comandi, digitando il carattere “/” e scegliendo fra le opzioni: “/tests”, per generare *test* di unità; “/fix”, per correggere errori di compilazione; “/explain”, per ricevere una descrizione testuale del codice.

Modellare le esigenze sopra delineate utilizzando un diagramma dei casi d’uso. Non è richiesta descrizione testuale del diagramma.

Risposta



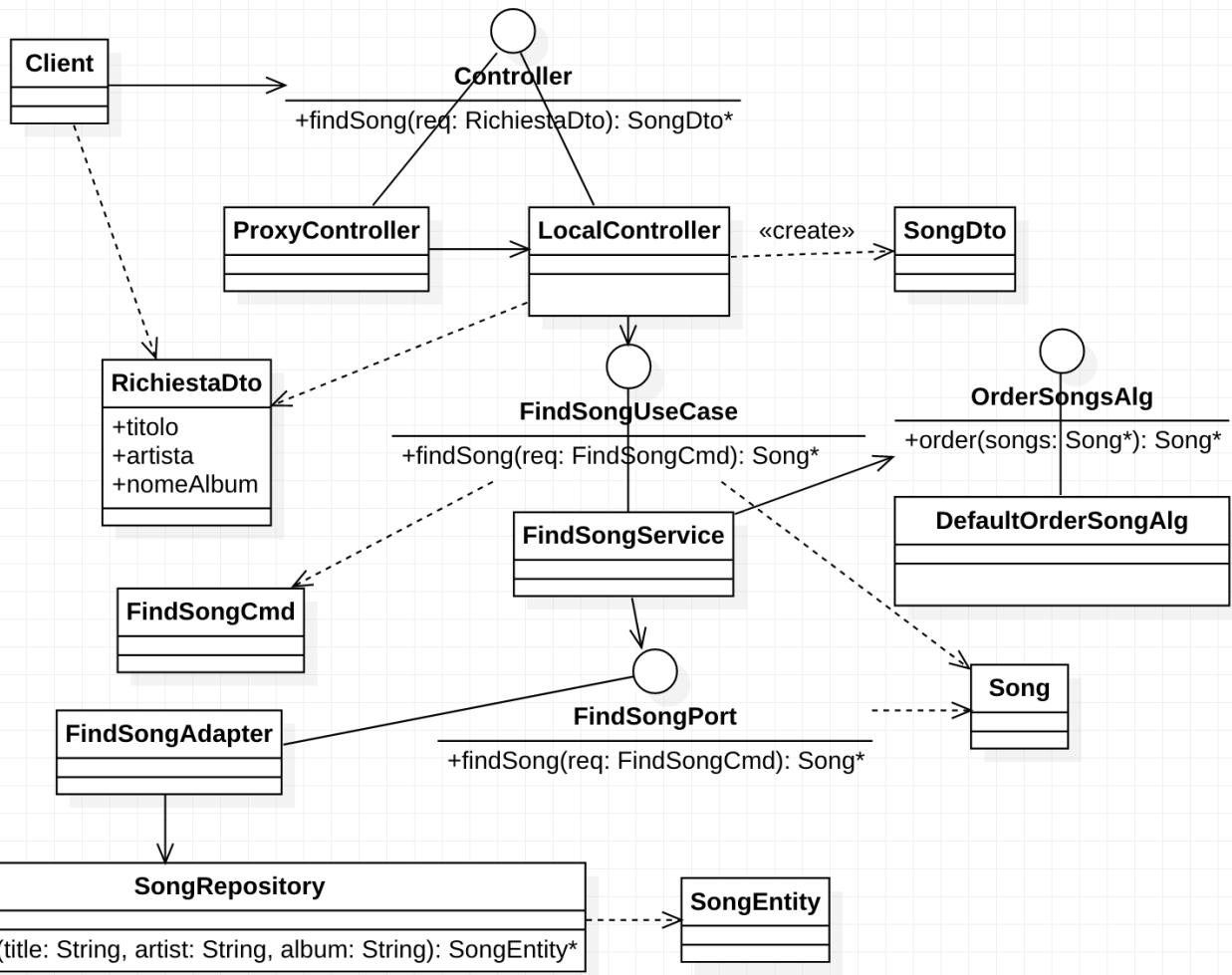
IstruzioniRiportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome: _____ Nome: _____ Matricola: _____ Anno Progetto Didattico: _____

Domanda 1/2 (punti 6/30)

Spotify è una nota piattaforma di *streaming* audio. L'applicazione è disponibile per diversi dispositivi. La sua versione *mobile* si connette a una delle istanze *server* così che l'utente possa richiedere l'ascolto di brani musicali. La richiesta contiene il titolo del brano e, optionalmente, l'artista e il nome dell'album in cui il brano è contenuto. Il *server* risponde con una lista di oggetti, corrispondenti ai risultati della ricerca nel *database* interno di *Spotify*. La comunicazione fra i due attori avviene come se essi eseguissero sul medesimo *host*. La ricerca viene gestita lato *server* con una classica architettura esagonale. Prima di essere inoltrati al *client*, i risultati vengono ordinati utilizzando un algoritmo intelligente, oggetto di continuo sviluppo, in quanto valore aggiunto competitivo del sistema.

Si modelli il sistema sopra descritto mediante un diagramma delle classi, comprensivo dei *design pattern* a esso pertinenti. Si presti particolare attenzione a dimostrare che l'architettura utilizzata sia effettivamente di tipo esagonale.

Risposta

Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

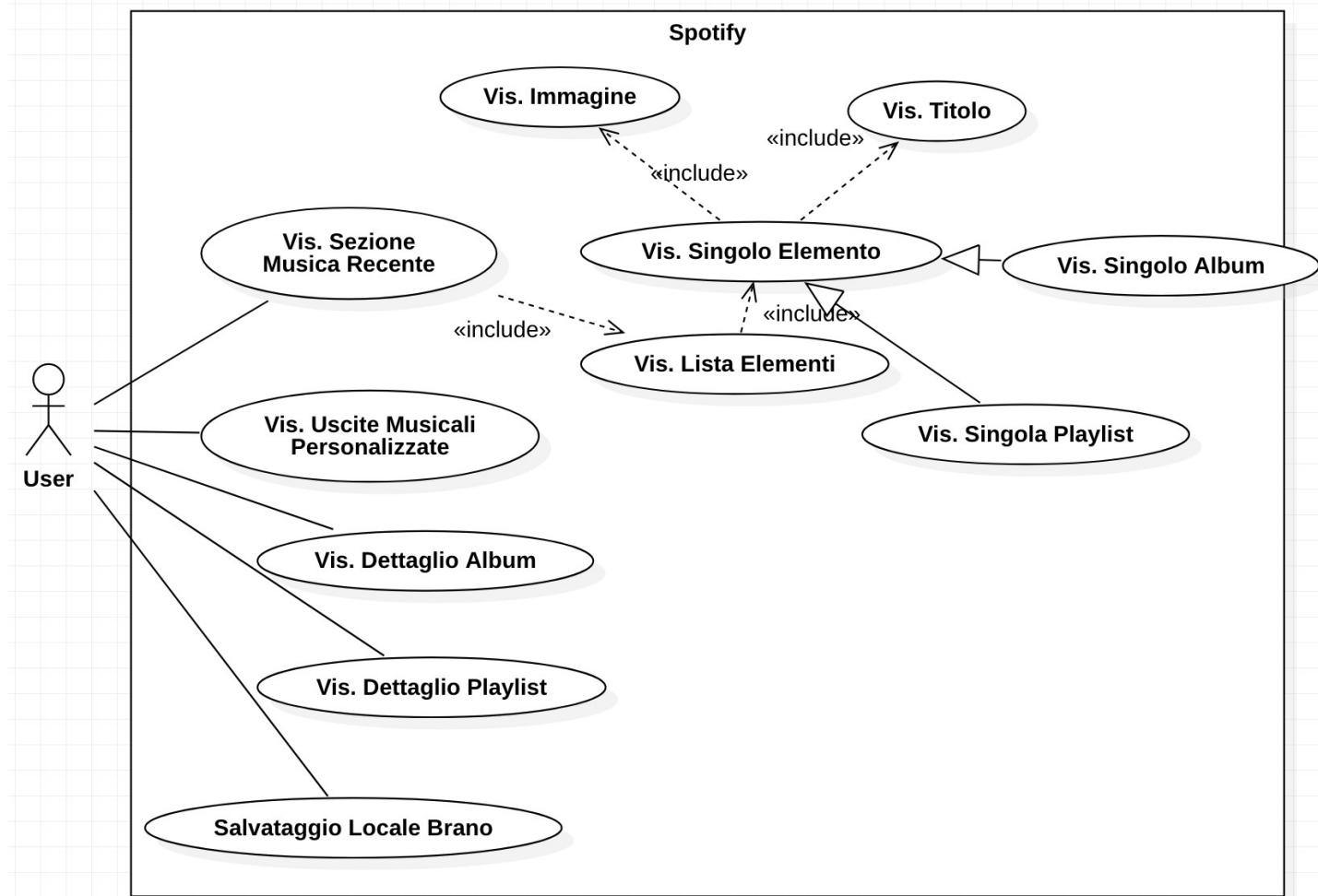
Cognome: _____ Nome: _____ Matricola: _____ Anno Progetto Didattico: _____

Domanda 2/2 (punti 4/30)

La pagina *home* della versione *mobile* dell'applicazione *Spotify* si presenta come un insieme di sezioni separate. Una sezione visualizza la musica ascoltata di recente, mentre un'altra le nuove uscite musicali, personalizzate per l'utente collegato. La prima delle due sezioni visualizza, per ogni oggetto nella lista, una immagine e un titolo. La selezione dell'oggetto porta l'utente in una sezione dedicata. Se l'oggetto rappresenta un album, ne vengono visualizzate la copertina, la durata intera dell'album, e la lista dei brani in esso contenuti. Nel caso in cui l'oggetto rappresenti una *playlist*, vengono visualizzati un'immagine dedicata, la lista di brani in lista, e il numero di salvataggi effettuati della *playlist*. In entrambi i casi, inoltre, anche i brani contenuti possono essere salvati in locale per un ascolto anche *offline*.

Modellare le esigenze sopra delineate utilizzando un diagramma dei casi d'uso. Non è richiesta descrizione testuale del diagramma.

Risposta



Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome:	Nome:	Matricola:	Anno Progetto Didattico:
----------	-------	------------	--------------------------

2024/2025

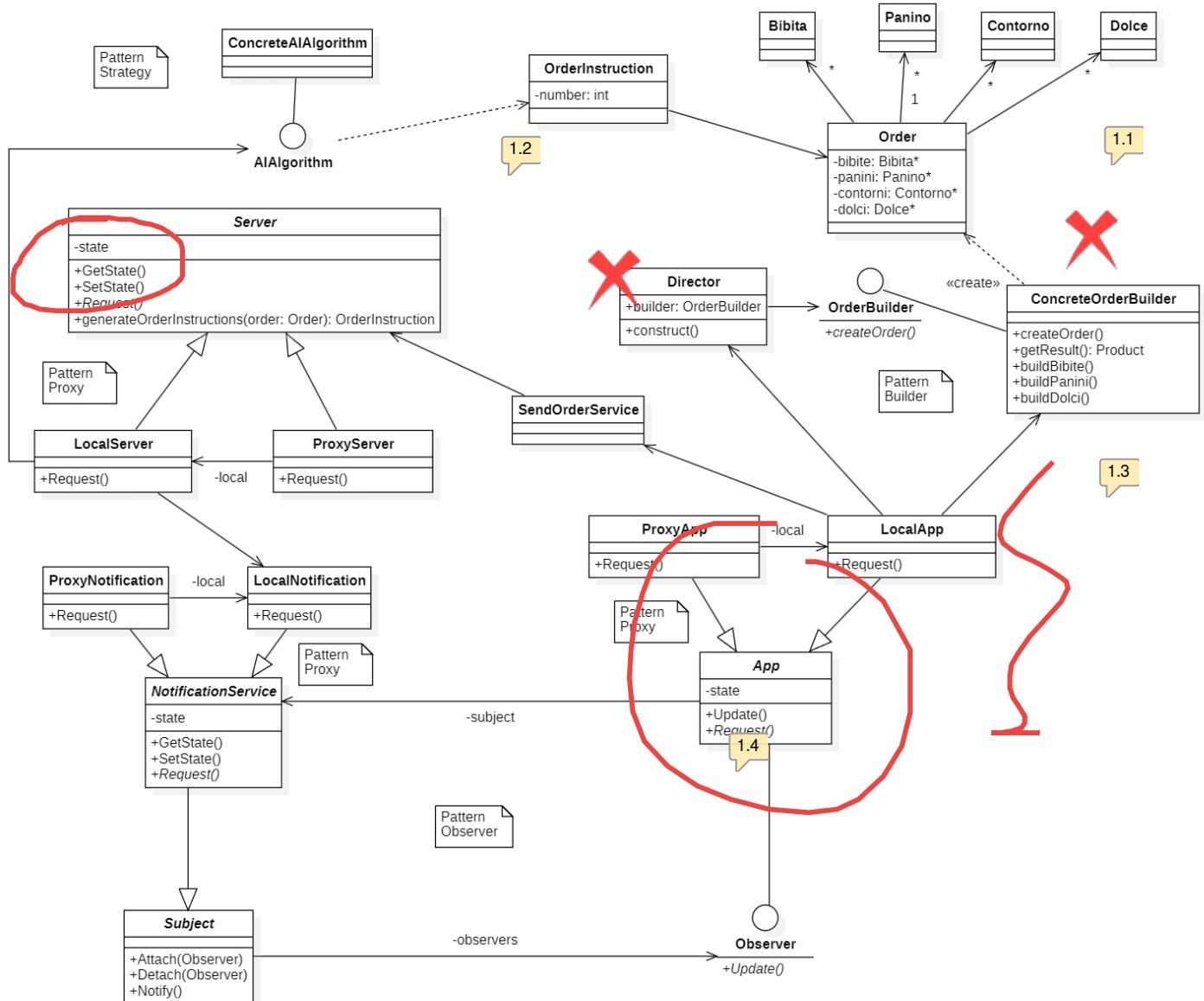
2024/2025

2024/2025

Domanda 1/2 (punti 6/30)

La catena di fast food *Vegburger Queen* ha sviluppato un'applicazione *mobile* per facilitare le ordinazioni in negozio. L'applicazione consente all'utente di creare il proprio ordine. L'ordine è rappresentato da un oggetto complesso, composto da una lista di bibite, una lista di panini, una lista di contorni, e una lista di dolci. Ogni "ordine" deve contenere obbligatoriamente almeno un panino; tutto il resto è opzionale. Una volta inviato l'ordine al *server* centrale, quest'ultimo lo elabora usando un algoritmo di intelligenza artificiale, che genera le istruzioni dettagliate per il servizio dell'ordine, associando anche un numero all'ordinazione, che viene ritornato all'applicazione *mobile*. L'algoritmo di IA migliora di versione in versione, senza però cambiare la propria interfaccia. Quando l'ordine è pronto, l'applicazione *mobile* riceve una notifica dal *server*, che invita il/la cliente a ritirare quanto ordinato, alle casse. L'applicazione *mobile* e il *server* dialogano tra loro come se entrambi stessero eseguendo in locale.

Si modelli tale sistema mediante un diagramma delle classi, comprensivo dei *design pattern* a esso pertinenti.

Risposta

Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome:	Nome:	Matricola:	Anno Progetto Didattico:
----------	-------	------------	--------------------------

2024/2025

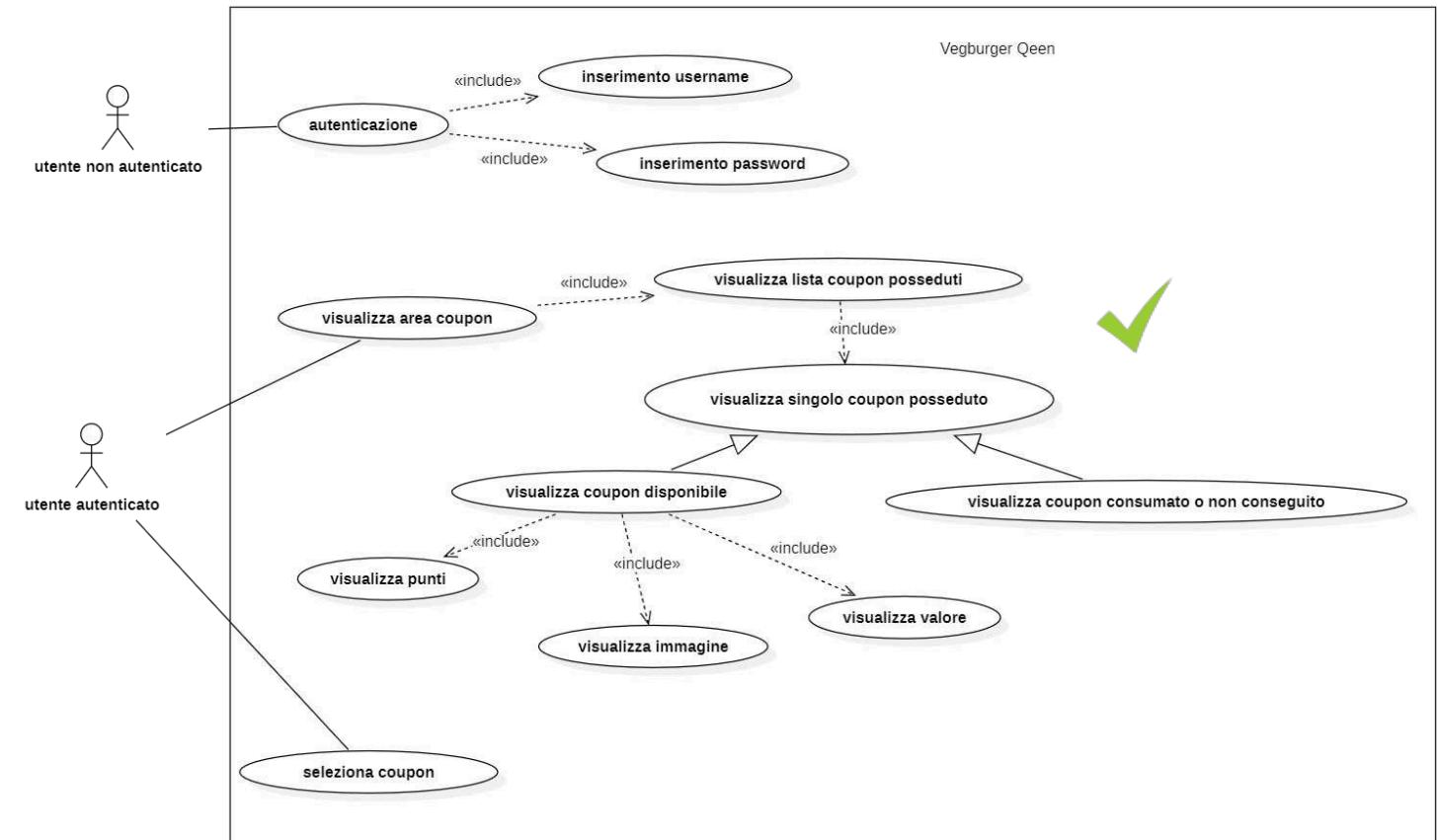
2024/2025

2024/2025

Domanda 2/2 (punti 4/30)

L'applicazione *mobile* della catena *Vegburger Queen* permette all'utente di verificare la disponibilità di *coupon* da utilizzare per ottenere sconti sul prossimo acquisto. Per accedere all'applicazione, l'utente deve autenticarsi con proprio *username* e *password*. Una volta riconosciuto/a, può accedere all'area *coupon* che visualizza come lista tutti quelli attualmente posseduti. I *coupon* disponibili sono ben visibili; gli altri (quelli già consumati o quelli non ancora completamente conseguiti) appaiono sfuocati. Per ogni *coupon* disponibile vengono visualizzati nella lista il numero di punti corrispondenti, un'immagine di cosa sia possibile acquistare con essi, e il suo controvalore in Euro. Per usare un *coupon* l'utente deve selezionarlo: in quel modo, al prossimo acquisto valido, al prezzo da pagare verrà automaticamente applicato lo sconto corrispondente.

Modellare le esigenze sopra delineate utilizzando un diagramma dei casi d'uso. Non è richiesta descrizione testuale del diagramma.

Risposta

Indice dei commenti

- 1.1 Che differenza c'è fra il metodo "getResult" e "createOrder"? Inoltre, le informazioni obbligatorie dovrebbero essere fornite al costruttore del builder
- 1.2 Il director come fa a sapere con quali informazioni costruire l'ordine?
- 1.3 Non sono chiari tutti questi metodo "Request()"
- 1.4 Sarebbe stato più opportuno individuare una parte dell'applicazione atta ad attendere la notifica.

Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

Cognome:	Nome:	Matricola:	Anno Progetto Didattico:
Picello Davide		2034825	2024/2025
Mahdi Mouad		2044222	2024/2025
Di Pietro Gabriele		2010000	2024/2025

Valutazione 7.5 /10

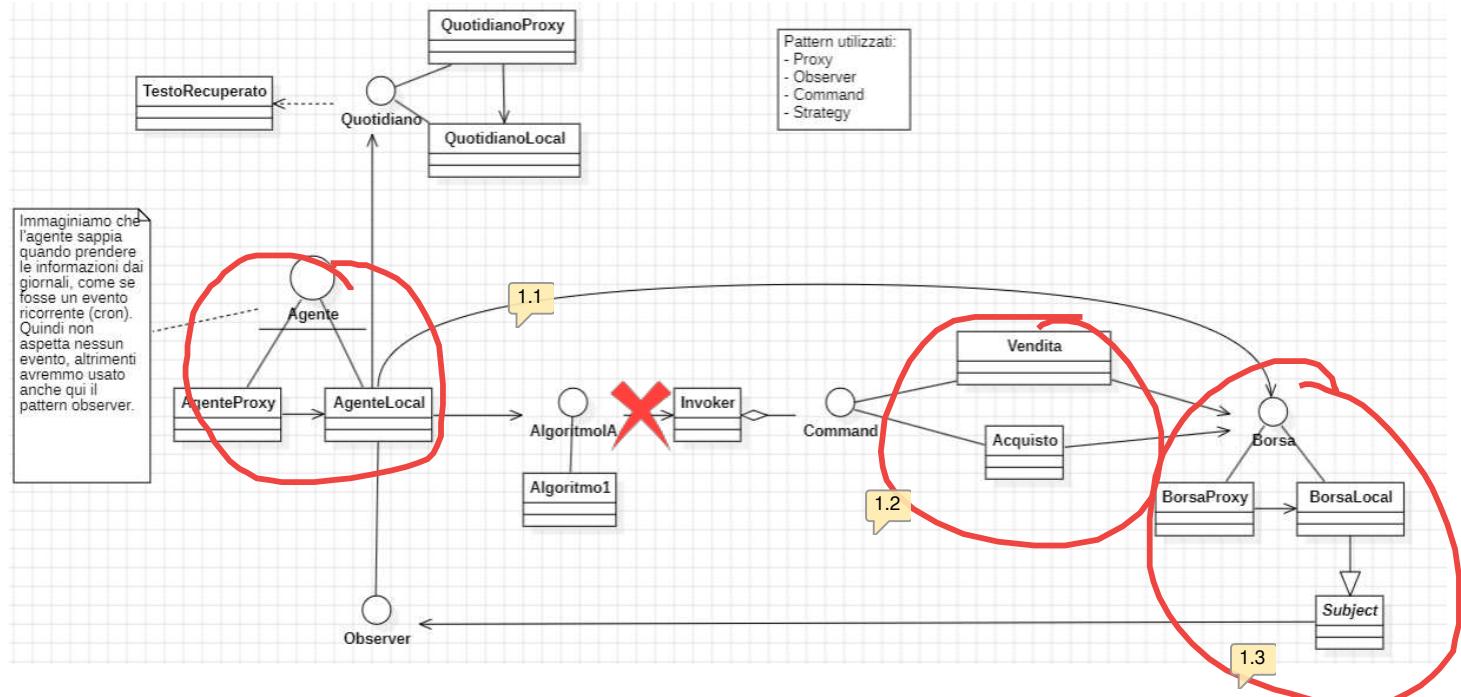
Domanda 1/2 (punti 6/30)

La carico perchè i casi d'uso sono 4/4

La tecnologia del momento è senza dubbio l'Intelligenza Artificiale di tipo *agentic*, cioè incorporata in agenti (applicazioni) *software* capaci di azione autonoma, come per esempio apportare modifiche a sistemi, recuperare informazioni da sorgenti esterne e rielaborarle per produrre risultati. L'azienda *Stock.ai* ha sviluppato una tecnologia di quel tipo per la compravendita di azioni a livello globale. Ll'agente di *Stock.ai* monitora le *news* sui siti di informazione internazionali più autorevoli, p.es., il *New York Times*. A intervalli regolari, l'agente effettua un *fetch* delle informazioni con un'operazione di GET sulla *home page* di uno specifico quotidiano. Il dialogo tra l'agente e il quotidiano avviene come se essi fossero entrambi locali l'uno all'altro. Successivamente, il testo recuperato viene fornito a un algoritmo di IA, che estrae una lista di coppie *stock/azione* dove l'azione è vendita o acquisto. Con queste informazioni, l'agente contatta il *server* della borsa di scambio valori opportuna e inizia le operazioni di compravendita. Nuovamente, tale comunicazione avviene simulando esecuzione locale. Le operazioni vengono effettuate in parallelo e in modalità asincrona. Il risultato di ogni singola operazione è osservato dall'agente; una volta disponibile, il risultato è fornito nuovamente all'algoritmo di IA, affinché esso aggiorni il suo modello interno a uso futuro.

Si modelli tale sistema mediante un diagramma delle classi, comprensivo dei *design pattern* a esso pertinenti.

Risposta



Istruzioni

Riportare qui sotto cognome, nome, matricola, e anno di progetto didattico di **tutti** i candidati che hanno collaborato alla risposta.

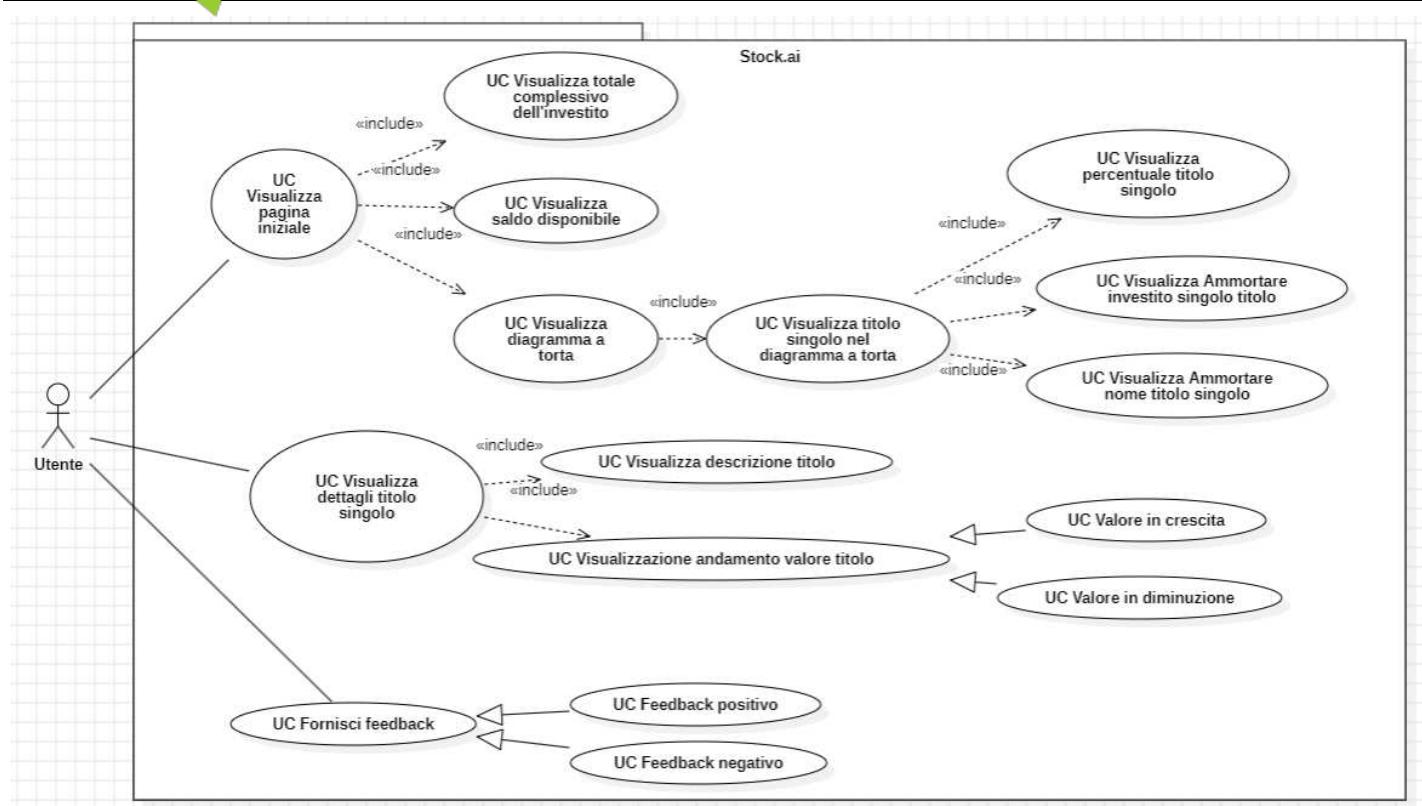
Cognome:	Nome:	Matricola:	Anno Progetto Didattico:
Picello Davide		2034825	2024/2025
Mahdi Mouad		2044222	2024/2025
Di Pietro Gabriele		2010000	2024/2025

Domanda 2/2 (punti 4/30)

Lo stato del sistema di *Stock.ai* può essere ispezionato dai propri utenti in ogni momento. La pagina iniziale visualizza il totale complessivo dell'investito e il saldo disponibile da investire. Essa inoltre visualizza un diagramma a torta che suddivide il saldo investito per titoli. Per ogni titolo è visualizzata la percentuale, l'ammontare investito e il nome. Selezionando un titolo dal diagramma a torta è possibile andare nel dettaglio dello stesso, dove viene riportata una sua breve descrizione narrativa, e se il suo valore sia in crescita o diminuzione. Inoltre, l'utente può fornire un *feedback* (positivo o negativo) sull'investimento, così da influenzare l'algoritmo di IA per le future operazioni.

Modellare le esigenze sopra delineate utilizzando un diagramma dei casi d'uso. Non è richiesta descrizione testuale del diagramma.

Risposta ✓



Indice dei commenti

- 1.1 GRAVE: Un'interfaccia NON può avere attributi. Inoltre, non è compito dell'algoritmo di AI di contattare il servizio esterno per comprare/vendere azioni
- 1.2 Non era richiesto alcun Command (comunque, non errato)
- 1.3 Il proxy e l'observer non si compongono così