Ingegneria del Software A.A. 2016/2017 Esame 2017-04-18

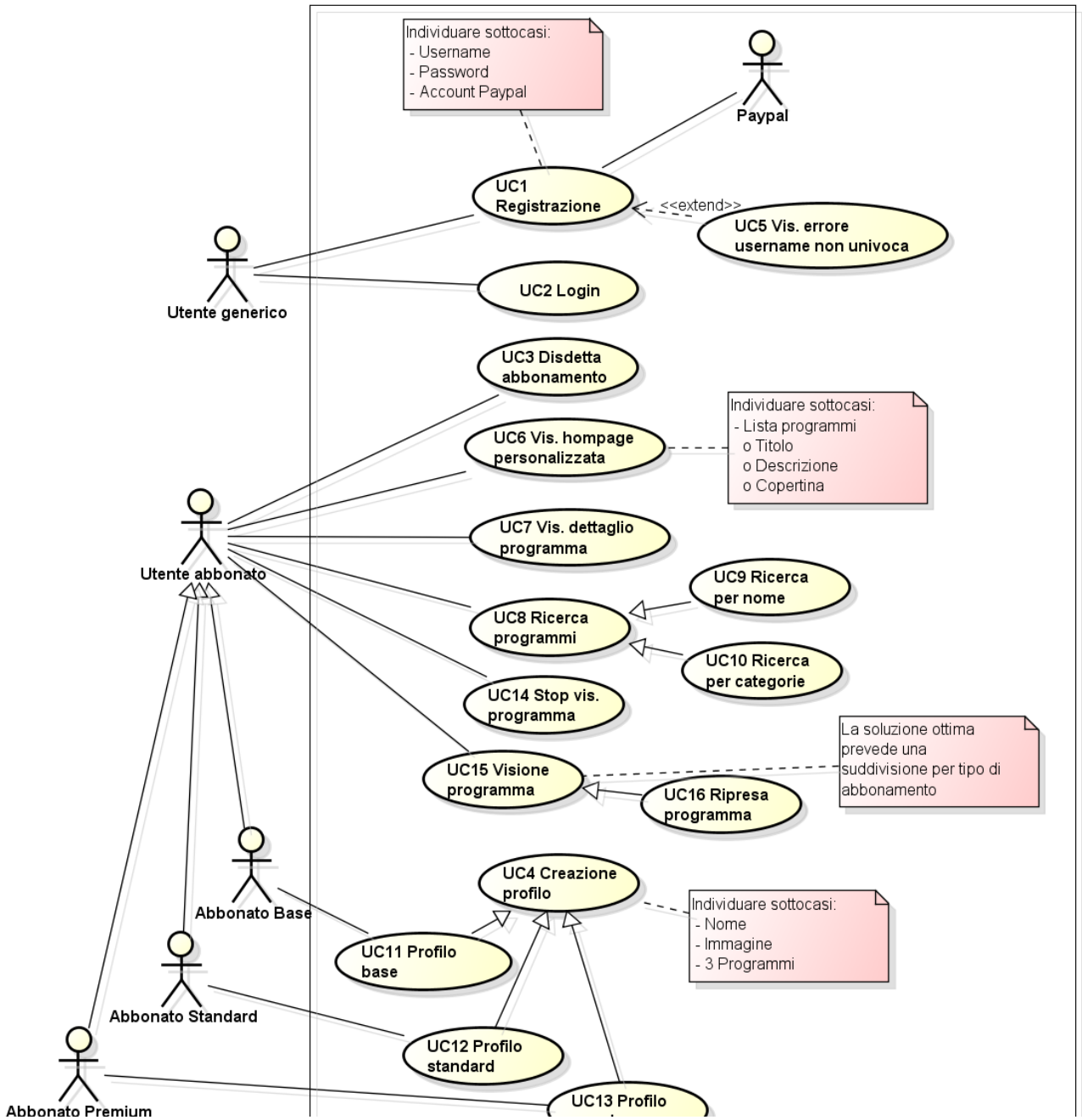
Esercizio 1 (6 punti)

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

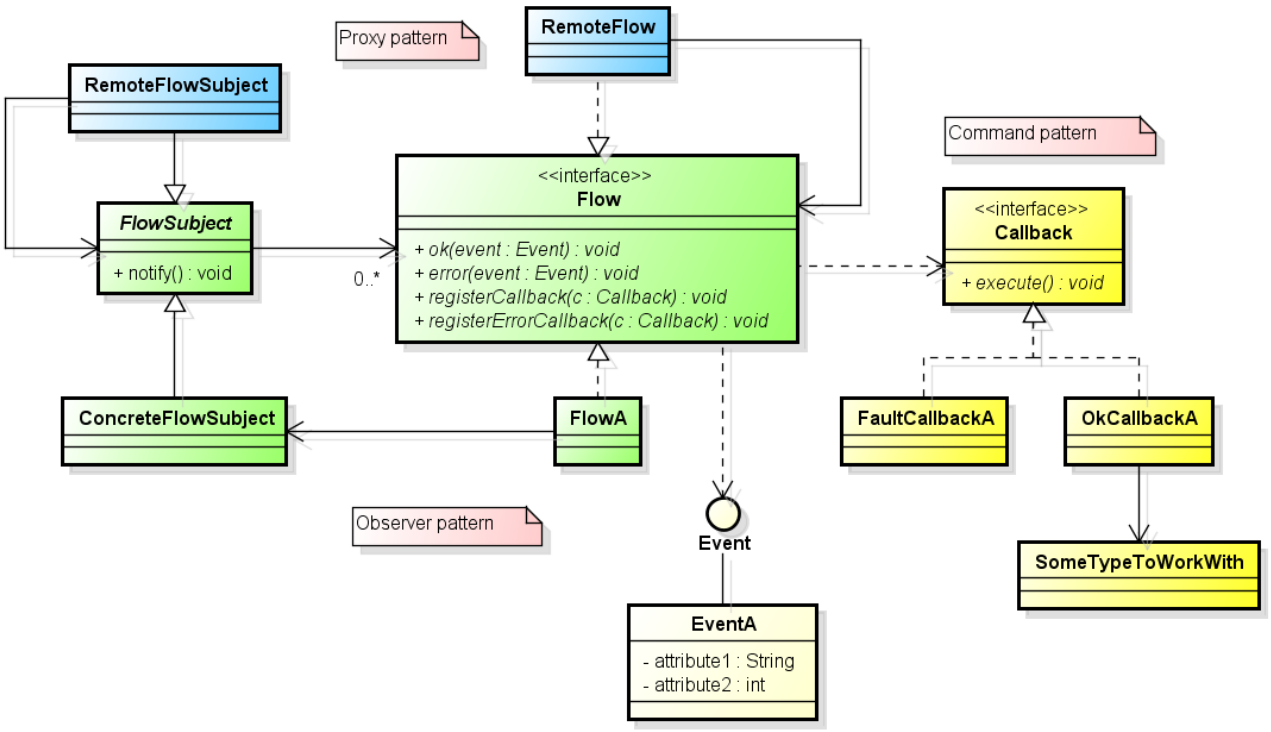
Da qualche anno si parla sempre più insistentemente di *reactive programming.* In questo stile di programmazione un oggetto non richiede ad un altro oggetto di eseguire delle operazioni, ma reagisce ad eventi. Esistono diverse librerie, che permettono di utilizzare questo stile di programmazione. Solitamente 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.

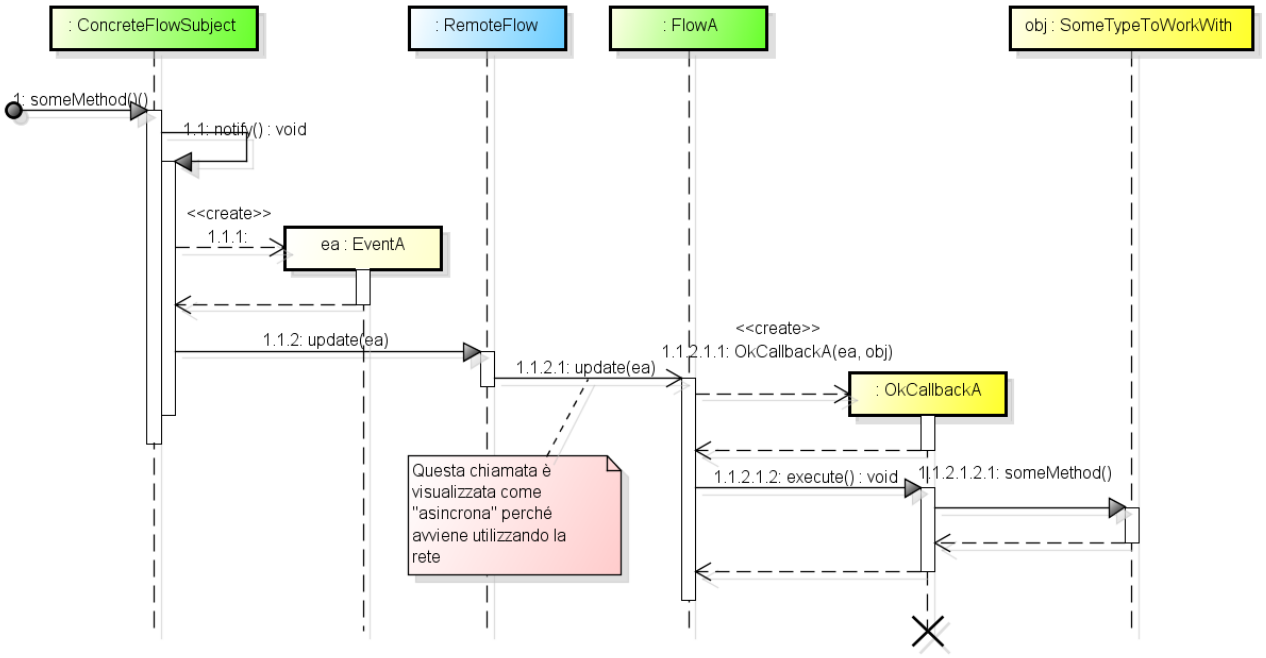
Si modelli il sistema descritto utilizzando un diagramma delle classi e gli opportuni *design pattern*. Inoltre, si descriva utilizzando un diagramma di sequenza la ricezione di un evento di tipo `EventA` da remoto, che scateni un'operazione generica attraverso una *callback* non di errore.

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 senguenti 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.