Ingegneria del Software A.A. 2017/2018 Esame 2018-07-20

Esercizio 1 (6 punti)

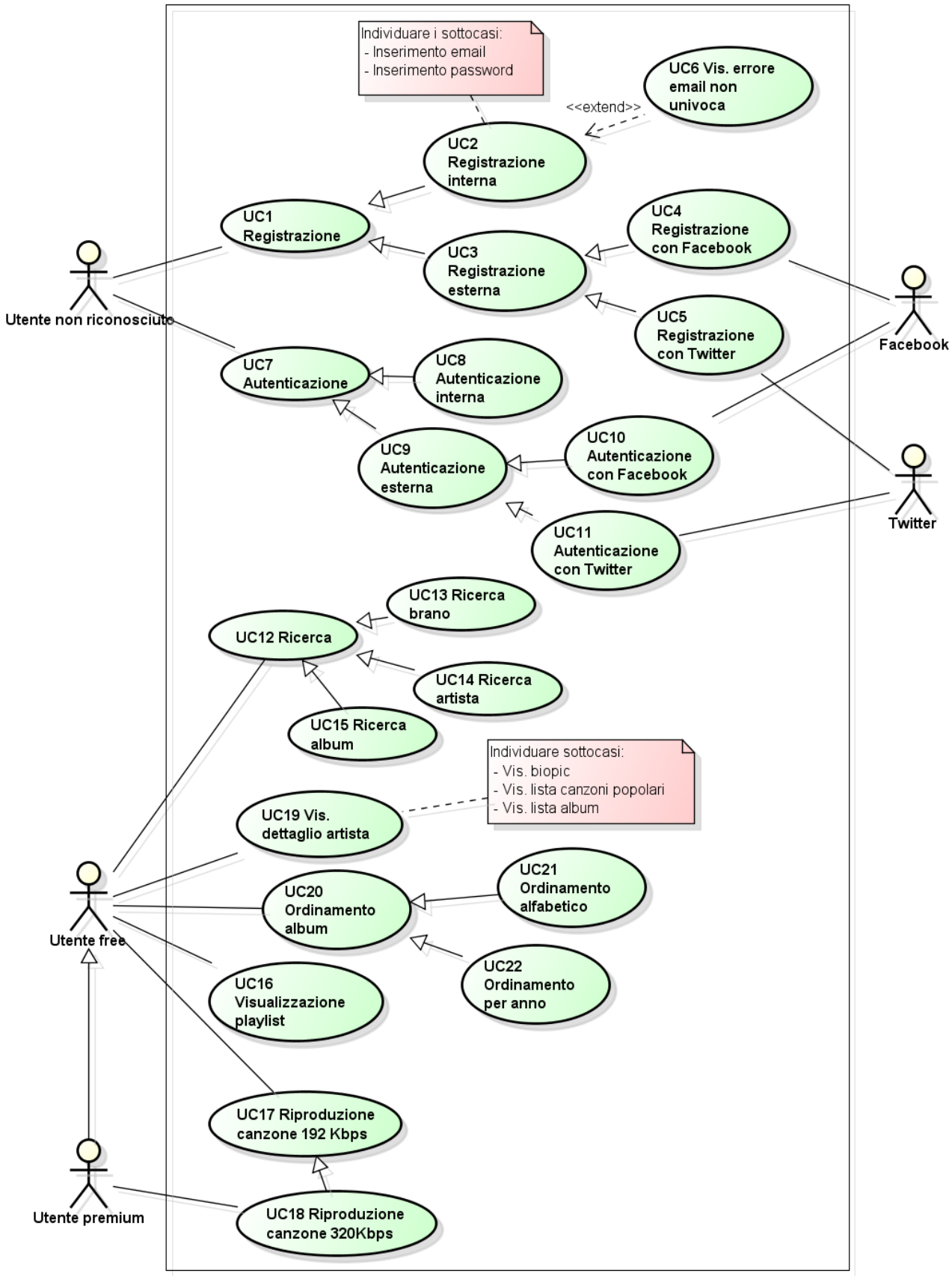
Descrizione

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.

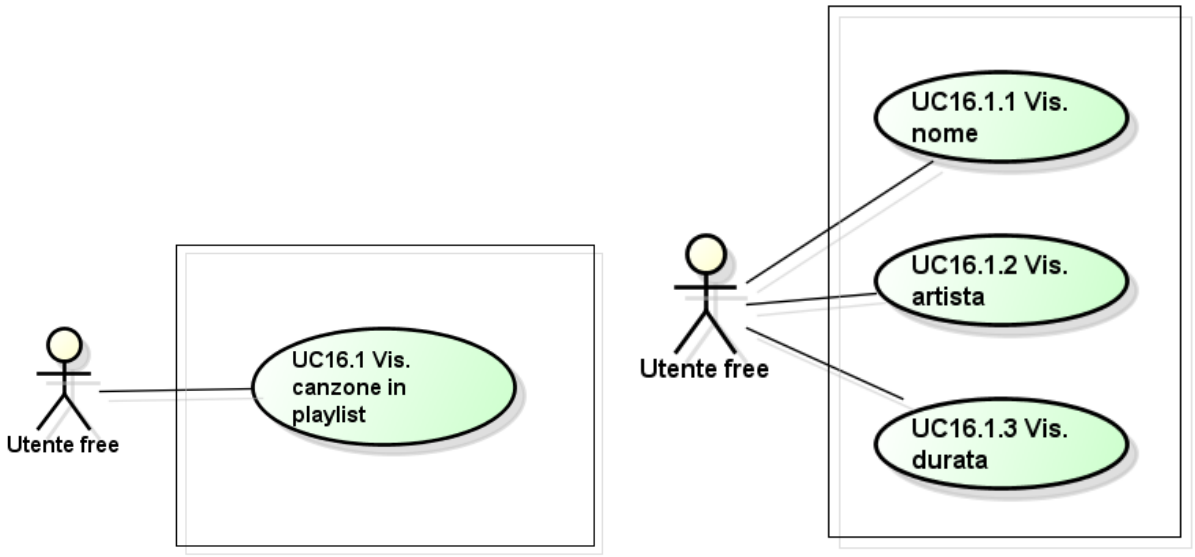
Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari sopra descritti. Non ne è richiesta la descrizione testuale.

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

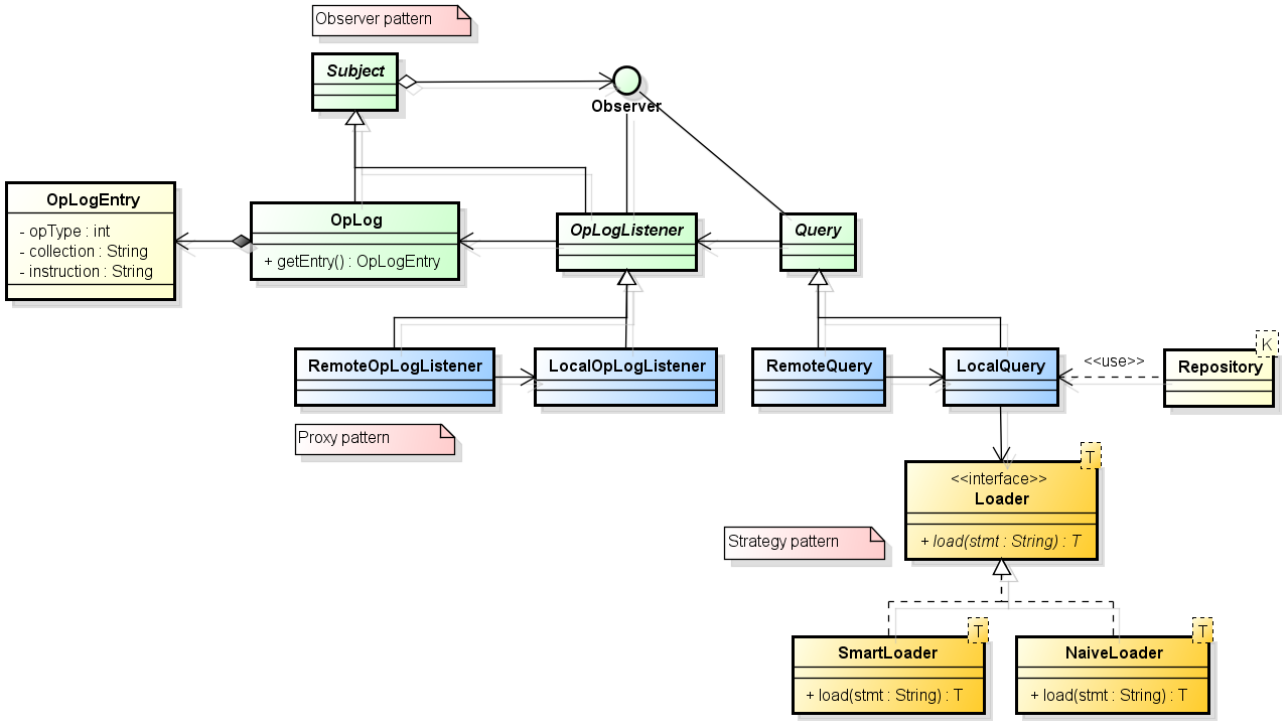
I nuovi driver di connessione a basi dati NoSQL e NewSQL sposano il paradigma *reactive* e il modello a eventi. Il *driver* `rx-mongo` per il DB NoSQL MongoDB è un esempio di questa tendenza. MongoDB è un DB documentale, che tratta i documenti come *file* JSON, e li indicizza con un campo riservato (*field*) 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*.

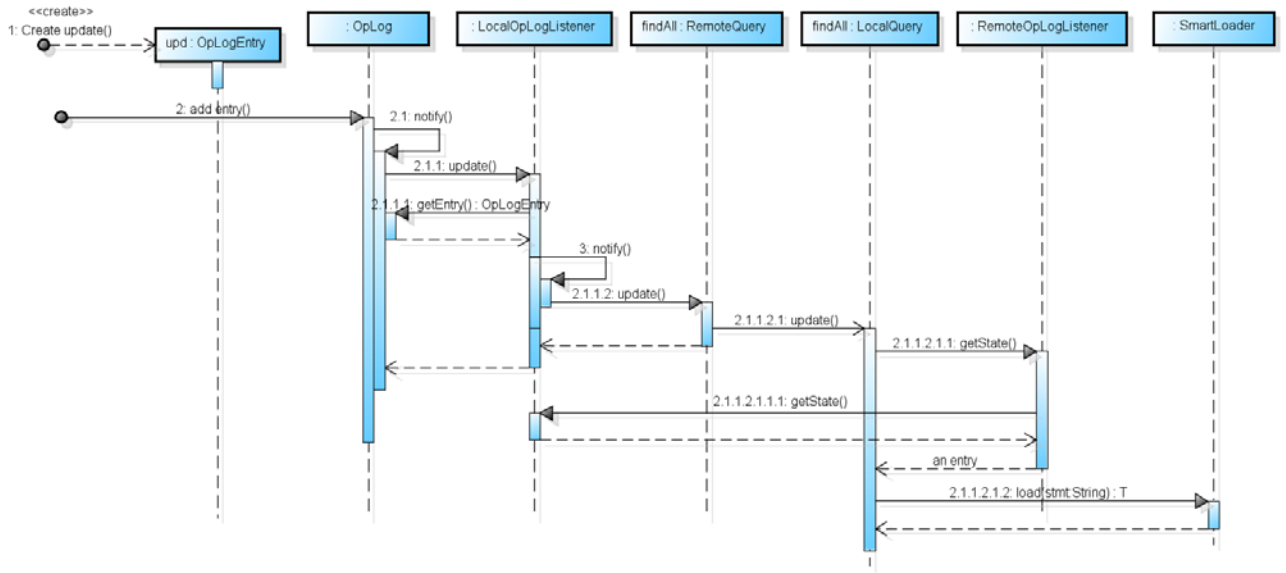
Soluzione

La soluzione prevede l'utilizzo dei seguenti design pattern:

- Observer pattern
- Proxy pattern
- Strategy pattern



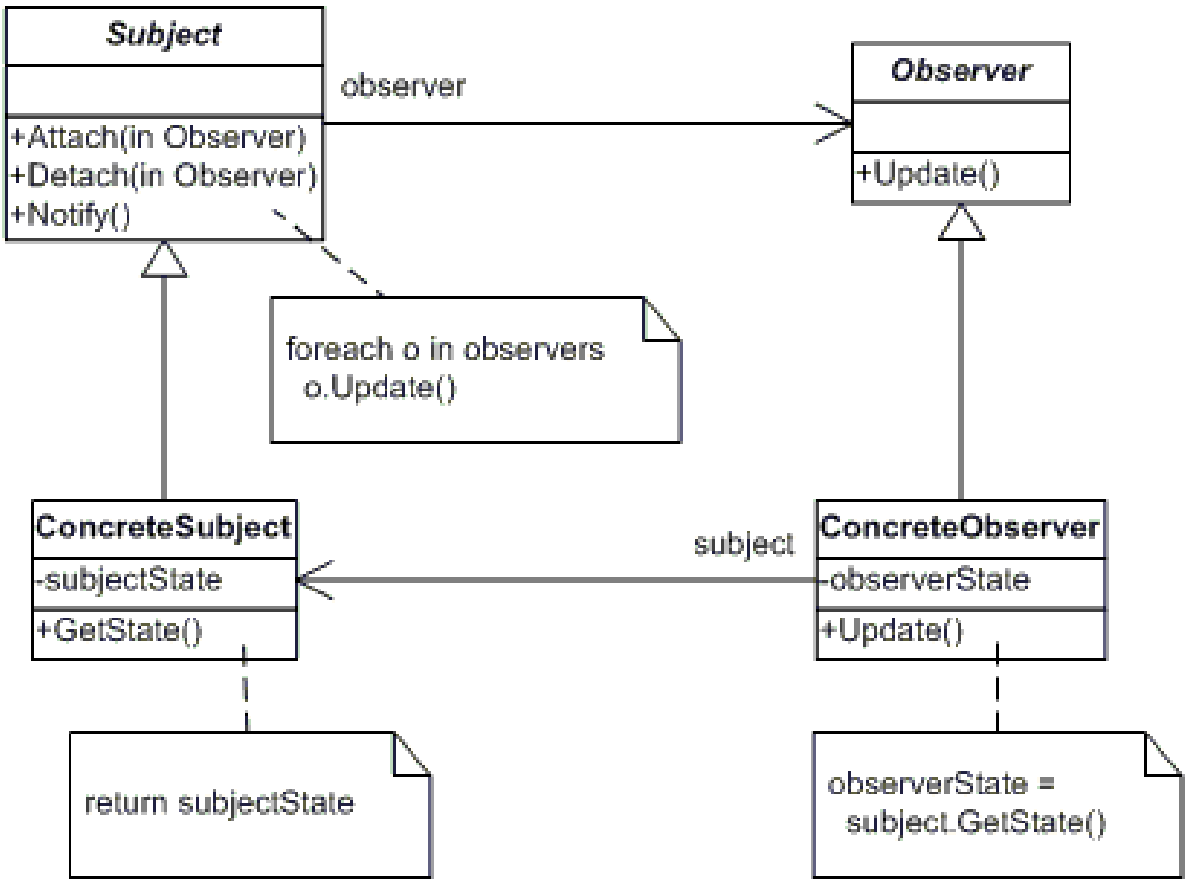
Il diagramma di sequenza richiesto è invece il seguente.



Esercizio 3 (3 punti)

Descrizione

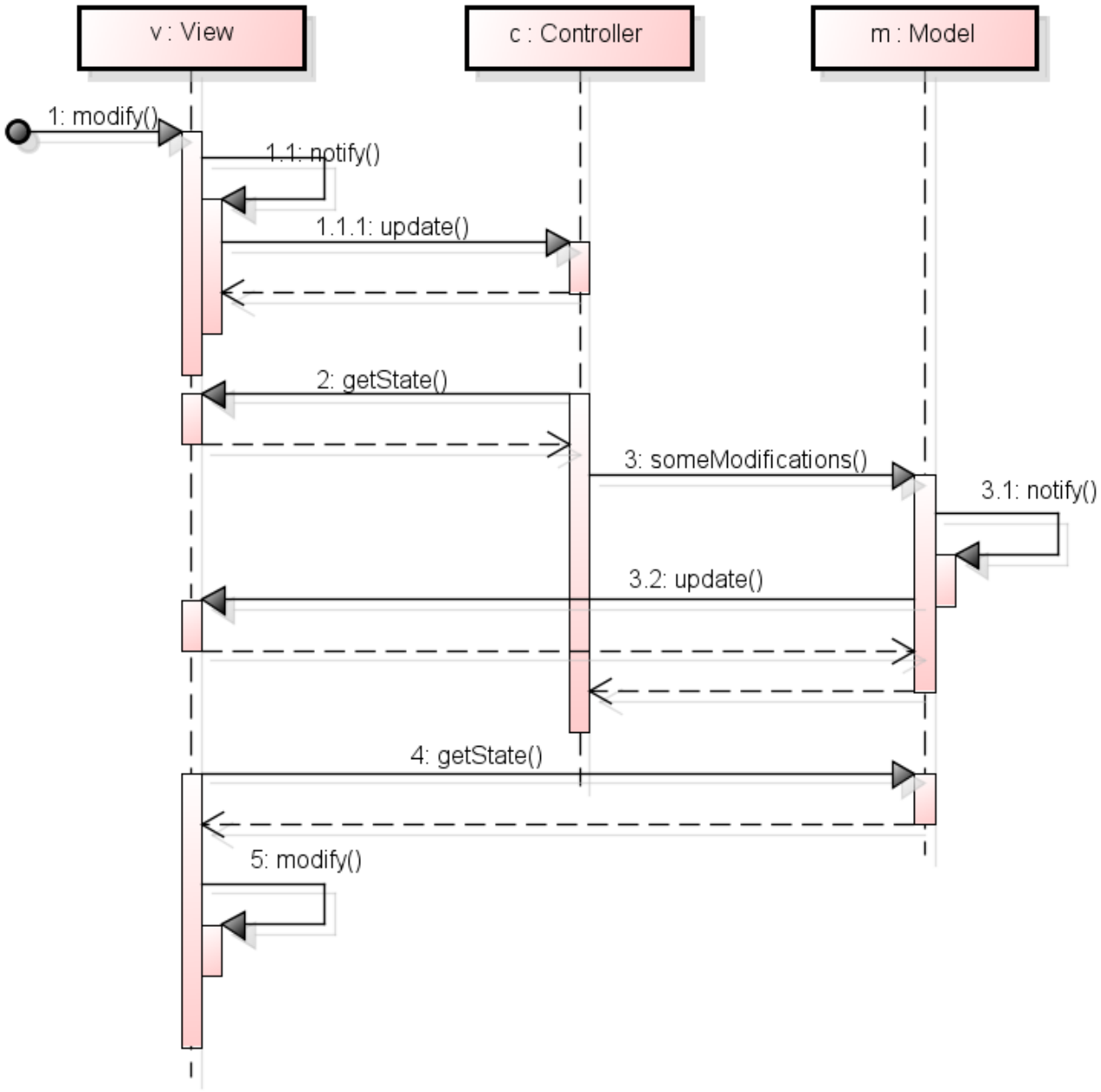
Nella sua versione base, nota come *push model*, il pattern Model View Controller, MVC, viene implementato utilizzando una doppia istanza del *pattern* Observer, rappresentato nel diagramma delle classi sotto riportato.



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.


v : View
c : Controller
m : Model
1: modify()
1.1: notify()
1.1.1: update()
2: getState()
3: someModifications()
3.1: notify()
3.2: update()
4: getState()
5: modify()