Ingegneria del Software A.A. 2017/2018 Esame 2018-08-24

Esercizio 1 (6 punti)

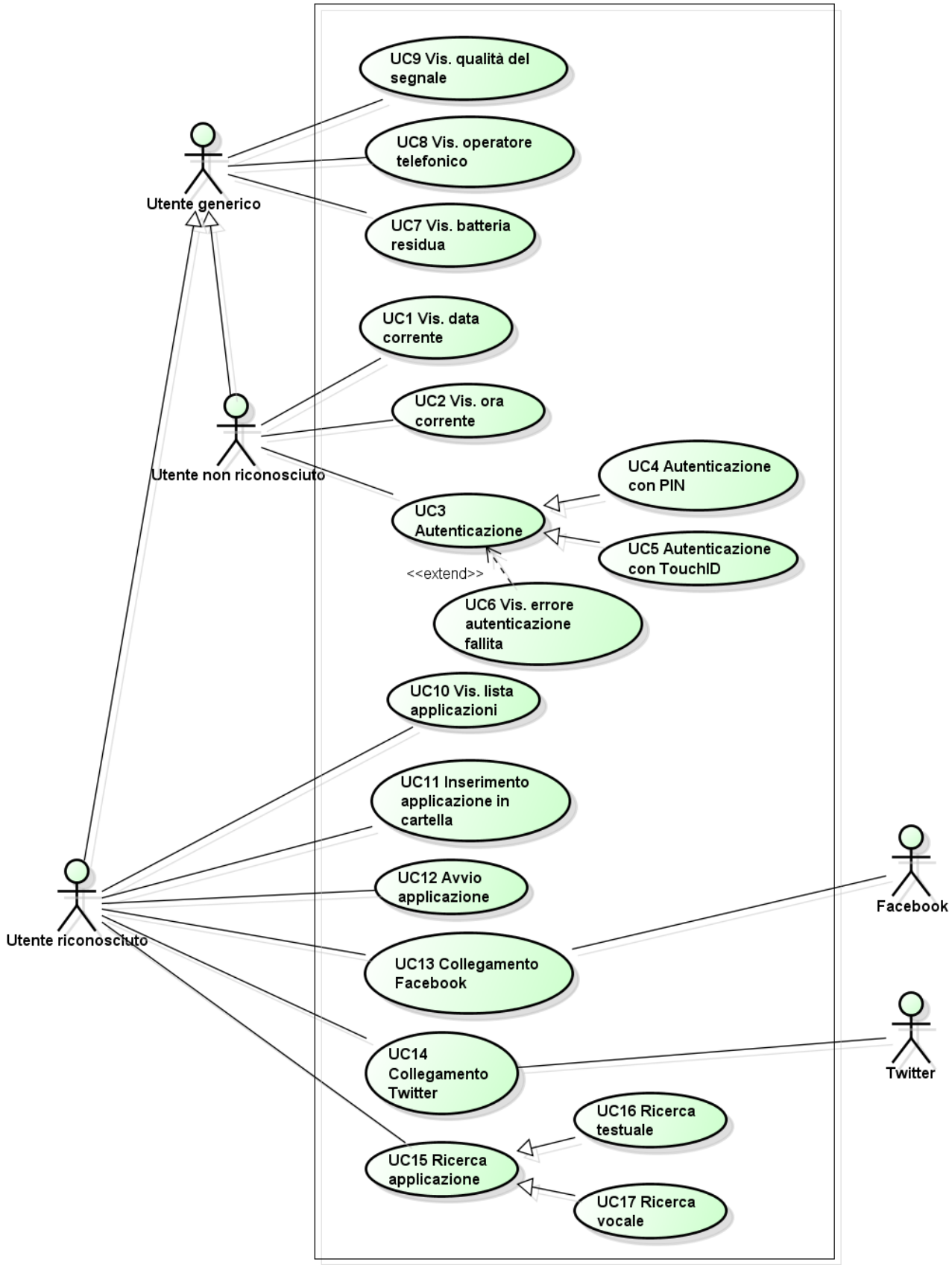
Descrizione

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.

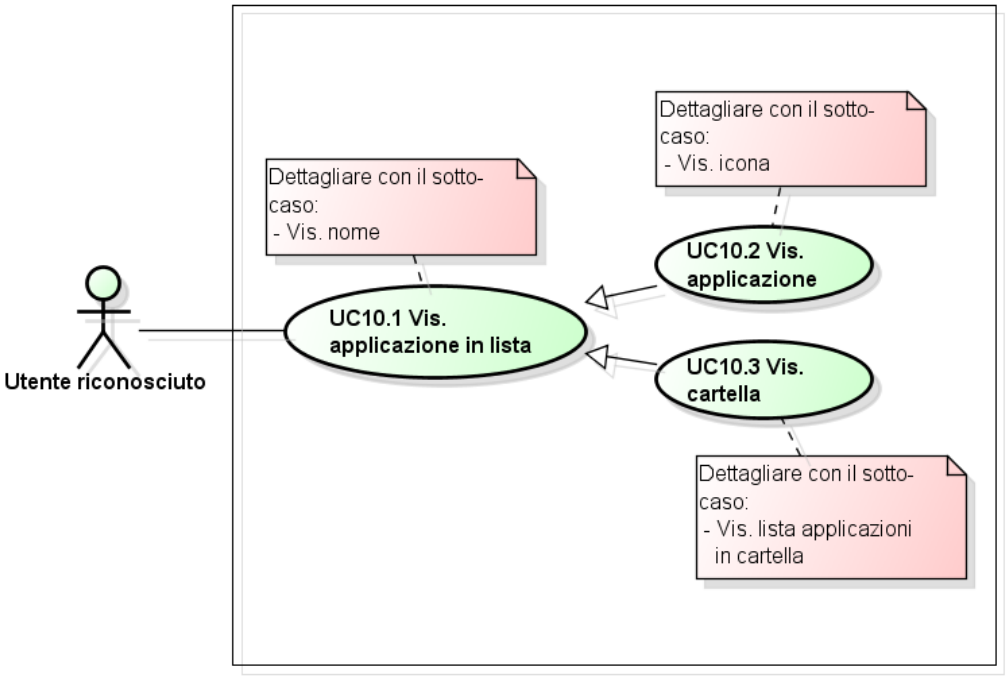
Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari sopra descritti. Non ne è richiesta la descrizione testuale.

Soluzione

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



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



Esercizio 2 (7 punti)

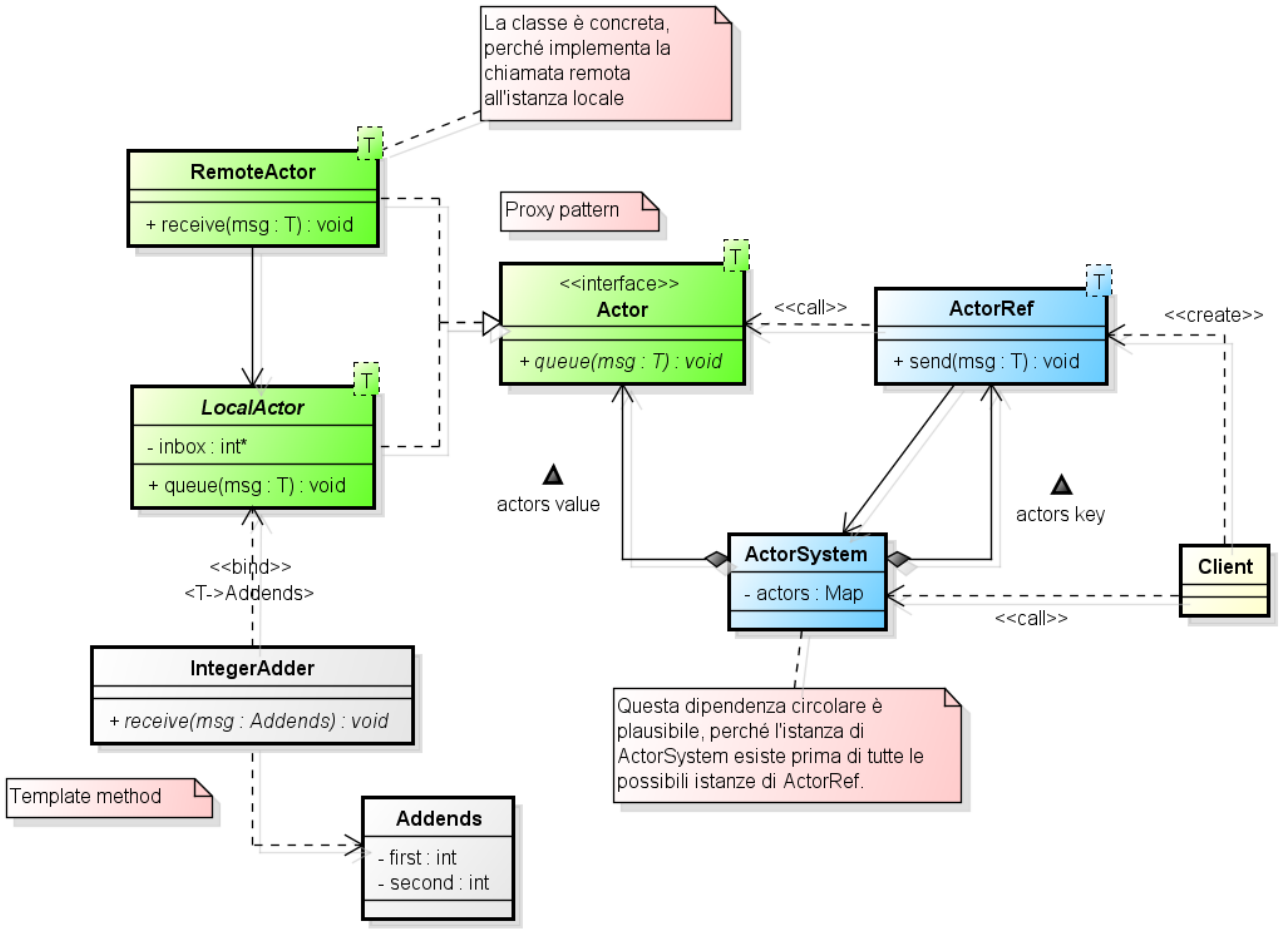
Descrizione

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*.

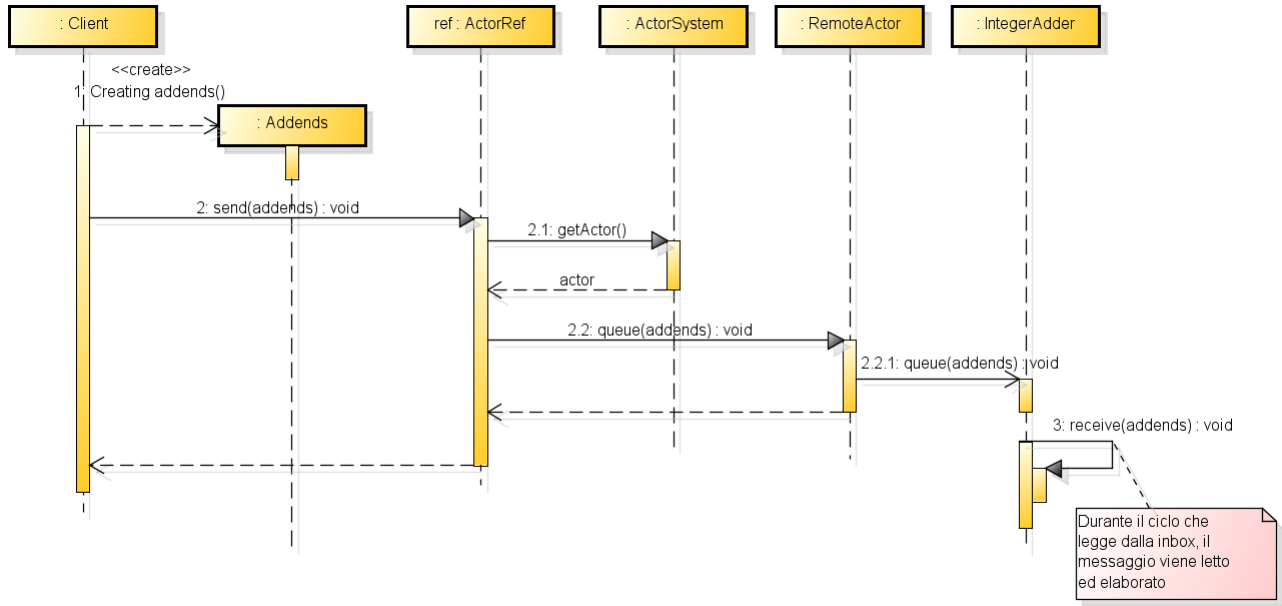
Si modelli tale sistema mediante un diagramma delle classi ed i *design pattern* a esso pertinenti. Utilizzando un diagramma di sequenza, si descriva l'invio di un messaggio contenente due interi ad un attore, che ne effettui la somma.

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.