Ingegneria del Software a.a. 2013-14
Bozza di soluzioni della Prova Scritta del 17 giugno 2014

Tempo a disposizione: 3 ore

Esercizio 1

Si consideri un distributore di merendine e bibite (prodotti nel seguito) automatico "di nuova generazione". Le azioni/eventi permesse/i all'utente del distributore sono le/i seguenti: **premuto tasto prodotto**, **immessa moneta** e **premuto tasto resto**. Il distributore è dotato di un visore che mostrerà all'utente il credito (ovvero il totale delle monete immesse). Se il credito è sufficiente e viene premuto il tasto relativo ad un prodotto allora il prodotto selezionato viene erogato (se invece il credito è insufficiente viene mostrato il prezzo del prodotto sul visore e viene richiesto l'inserimento di altre monete). Il tasto resto viene utilizzato per recuperare il resto (quando presente nel distributore) o le monete inserite. Rispetto ai distributori "di vecchia generazione" questo distributore è anche in grado di notificare al fornitore, mediante invio di sms, la mancanza di un prodotto quando la sua disponibilità è pari a zero.

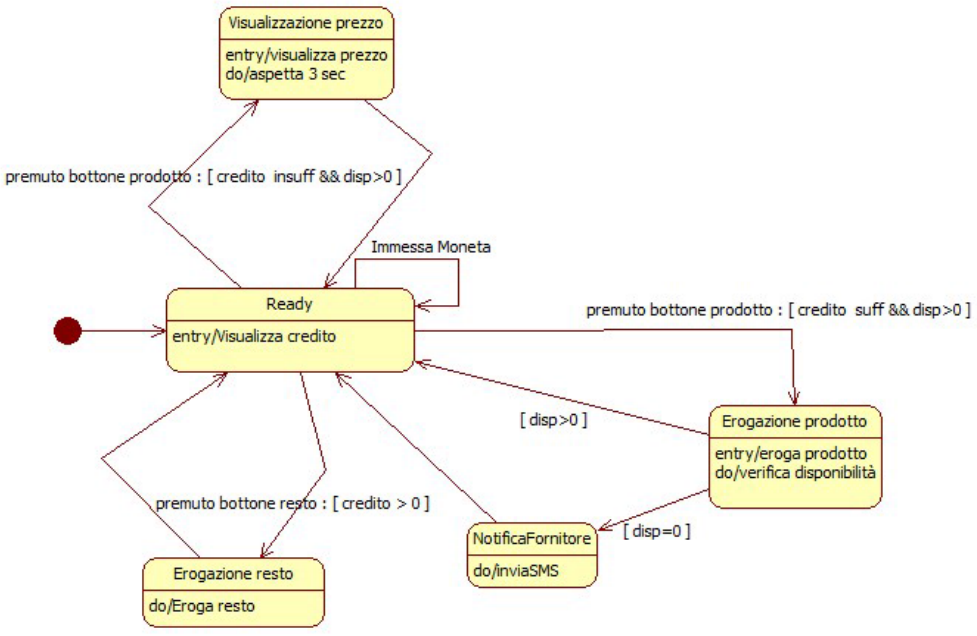
a) Quale tra i diagrammi UML visti a lezione è il più indicato per modellare il distributore descritto sopra e perchè?

*Il diagramma più indicato per modellare il distributore descritto sopra è lo **UML state diagram** perché è un diagramma comportamentale in grado di modellare un sistema che cambia il suo stato interno a seconda delle azioni intraprese dall'utente.*

b) In quanti e quali stati si può trovare il distributore di merendine e bibite?

E' possibile modellare il distributore descritto sopra utilizzando i seguenti 5 stati: Ready (che rappresenta lo stato principale nel quale l'utente può eseguire le tre azioni sopraccitate), Visualizza prezzo, Erogazione resto, Notifica fornitore, Erogazione prodotto.

c) Modellare tramite un diagramma UML il comportamento del distributore (gli eventi permessi all'utente del distributore dovranno comparire in modo esplicito nel diagramma)

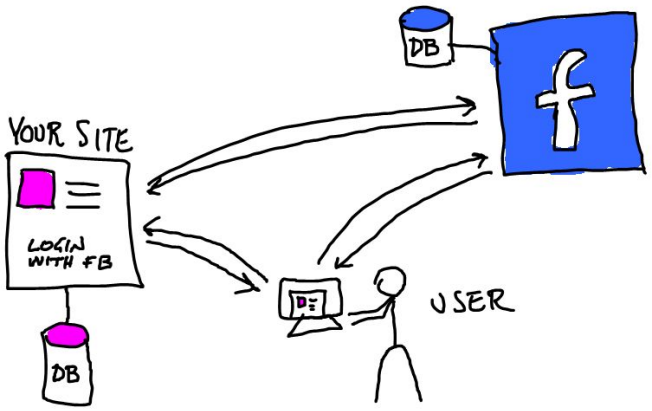


d) Quale tipo di design pattern ritenete sia utile per implementare il sistema di gestione del distributore e perchè?

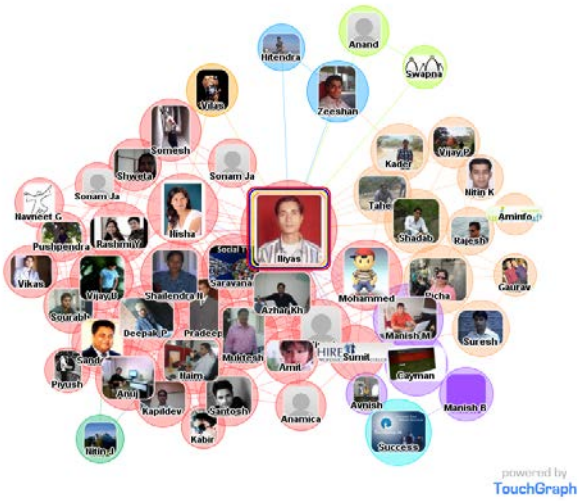
Design pattern State perché permette di implementare in modo semplice ed elegante la state diagram mostrato sopra.

Esercizio 2

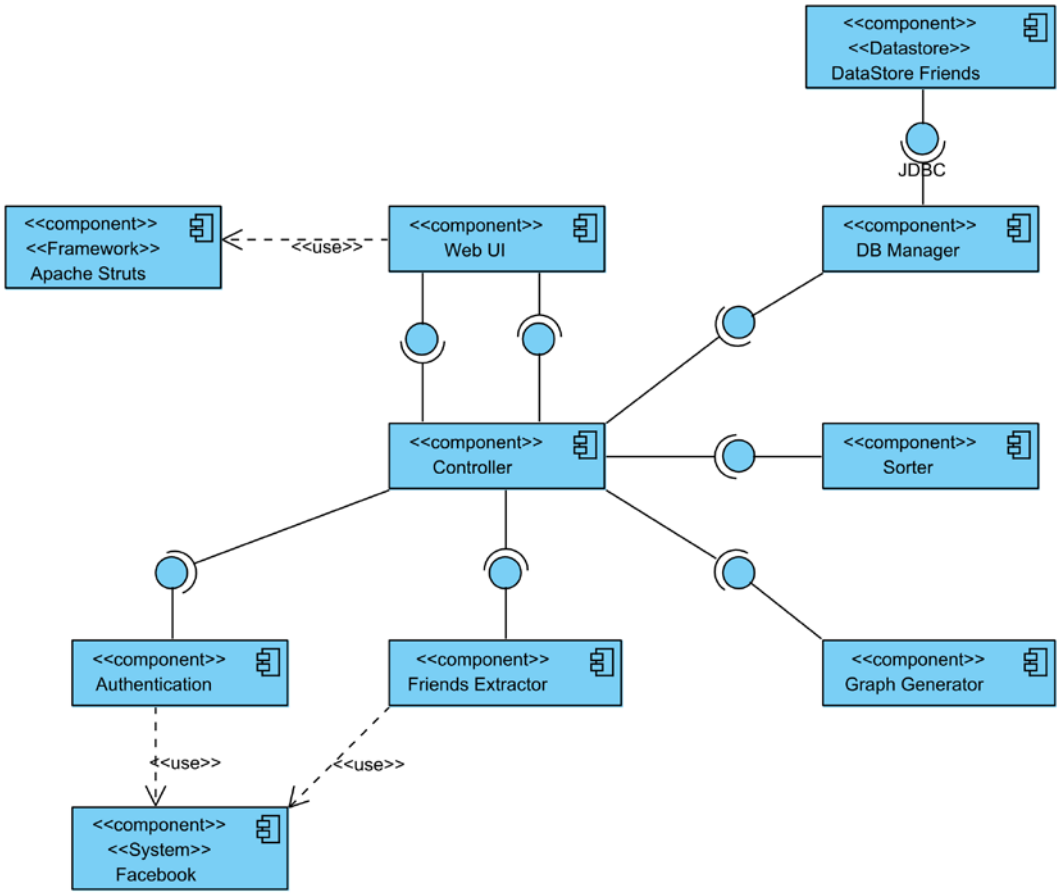
Si supponga di dover sviluppare un applicazione Web di **Facebook analytics** chiamata **FAnalytics**. Tale applicazione Web deve permettere all'utente di effettuare la **login** su Facebook. Una volta che l'utente è loggato l'applicazione Web deve essere in grado di recuperare la lista degli amici, visualizzarli in modo ordinato mediante una lista in una pagina Web e registrarli in un DB locale. La figura successiva da un idea molto generale di cosa si vuole realizzare.



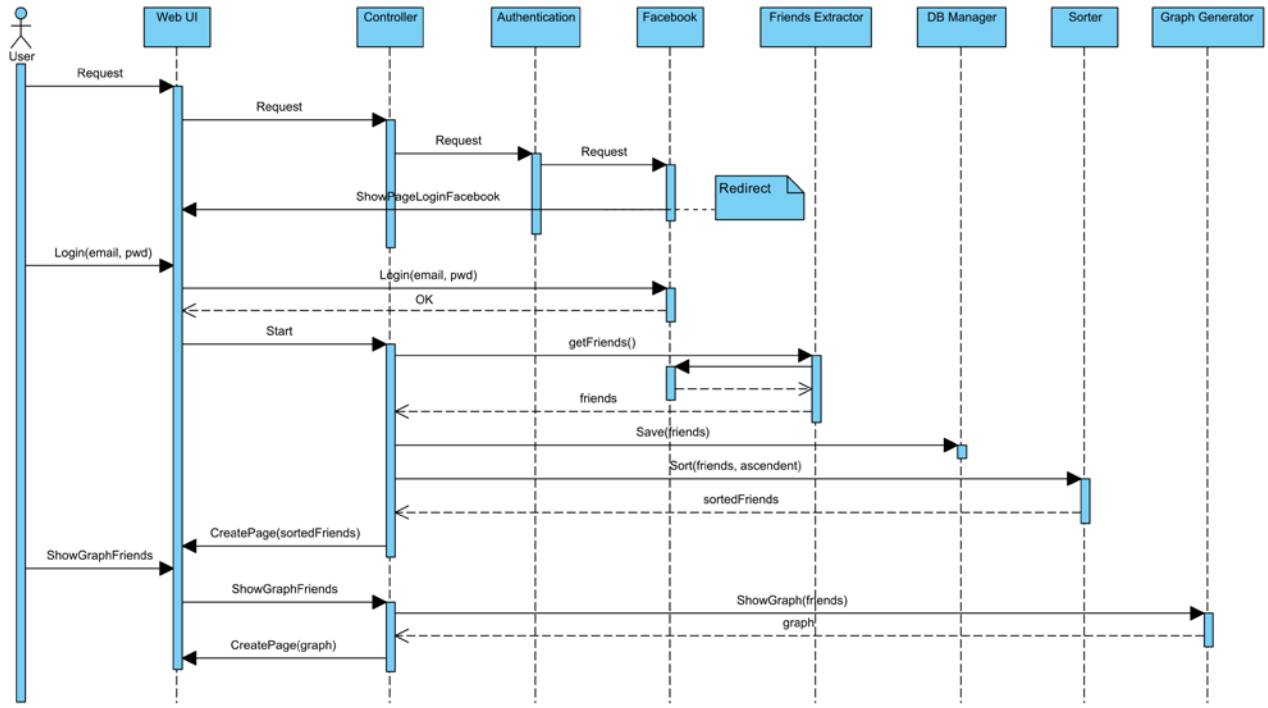
A questo punto l'utente può decidere di visualizzare il grafo delle amicizie premendo un apposito bottone che farà comparire una finestra pop-up contenente il grafo. Vedere la figura successiva.



a) Supponendo di avere già a disposizione le componenti software per effettuare il Login Facebook (*Login component*), recuperare gli amici (*GetFriends component*) e creare il grafo (*CreateGraph component*) rappresentare l'architettura di una possibile applicazione **FAnalyics** mediante un component diagram UML

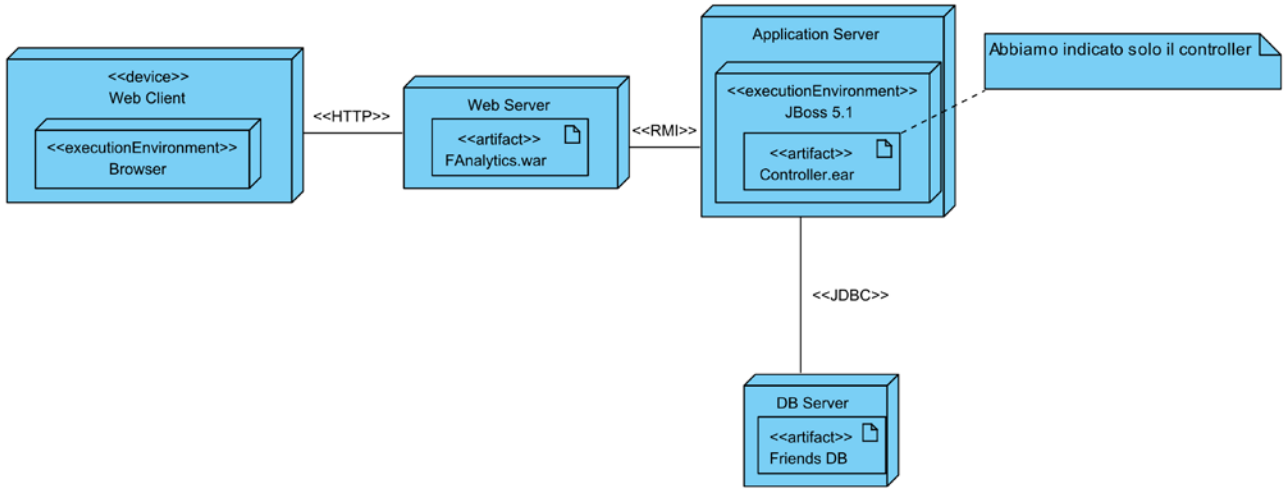


Il funzionamento si può intuire guardando il seguente sequence diagram:



b) Mostrare una possibile architettura hardware per l'applicazione **FAnalyics** utilizzando un diagramma UML che ritenete opportuno

Deployment diagram:



Esercizio 3

Dato il seguente metodo Java:

```
public static int[] foo(int[] a) {
          if (a == null)
               return null;
          else {
               int tmp;
               for (int i = 0; i < a.length - 1; i++) {
                    for (int j = 0; j < a.length - 1 - i; j++) {
                         if (a[j + 1] < a[j]) {
                              tmp = a[j];
                              a[j] = a[j + 1];
                              a[j + 1] = tmp;
                         }
                    }
               }
               return a;
          }
     }
}
```

a) Specificare in linguaggio naturale o OCL precondizioni e postcondizioni del metodo **`foo`**

Precondizioni:nessuna. ***`a!=null`*** *non serve perché il controllo viene effettuato all'interno del metodo.* ***`a.length >= 1`*** *non serve perché il metodo funziona correttamente anche con array vuoto (il corpo del for esterno non viene mai eseguito).*

Postcondizione:

```
(a==null and result==null) or ((result.length==a.length) and
forall (i:Integer | (0<=i and i<a.length) implies
result[i+1]<=result[i]))
```

b) Quale è la complessità ciclomatica del metodo **`foo`**? Spiegare in modo conciso come è stato svolto il calcolo

5, ottenuto come 4 + numero di if e for nel metodo

c) Quali casi di test ritenete sia oppurtuno considerare per testare il metodo? Spiegare le scelte e fare degli esempi

```
a = null
a = {}
a = {5}   // casi limite su dimensione array
a = {5,8,12}  //array già ordinato
a = {5,8,0,2,1}
a = {5,8,8,5,-1} // due elementi uguali
```

d) Quali criteri di copertura sono soddisfatti dai casi di test in c)? Giustificare la risposta

Statement (node) coverage: la test suite esegue tutti i comandi almeno una volta

Branch coverage: la test suite percorre tutti i branch almeno una volta (per ogni for e if si percorrono ramo true e ramo false almeno una volta)

Multiple Condition Coverage (MCC): coincide con branch coverage, non ci sono condizioni multiple

All Paths coverage: no, abbiamo problema dei loop. Neanche con semplificazione il loop non è eseguito/il loop è eseguito una volta: per come è scritto, il for interno è sempre eseguito almeno una volta se si entra nel for esterno