Università degli Studi di Roma "La Sapienza"
Facoltà di Ingegneria – Corso di Laurea in Ingegneria Gestionale
Corso di Progettazione del Software
Proff. Toni Mancini e Monica Scannapieco

Progetto **PC.20070711**

versione del 11 luglio 2007

Si vuole progettare e realizzare *PizzaExpress*, una applicazione informatica che consenta ad una pizzeria di ricevere prenotazioni di pizze, bevande ed altre specialità, e di effettuare le relative consegne ai clienti per mezzo di pony-express dotati di scooter.

Si richiede di effettuare la fase di Analisi, producendo uno schema concettuale per l'applicazione i cui requisiti sono descritti in calce.

L'applicazione *PizzaExpress* deve consentire ai clienti di una pizzeria di effettuare ordini via web di pizze, bevande, ed altre specialità. In particolare, collegandosi al sito web della pizzeria, i clienti possono consultare la lista delle diverse specialità disponibili, con il relativo prezzo. Le voci della lista sono divise per categorie (ad es., pizze, antipasti, fritti, bevande, dolci, ecc.).

Il sistema deve permettere ai clienti di effettuare ordini, dichiarando anche l'indirizzo e l'ora indicativa di consegna, oltre che gli estremi della loro carta di credito (tipo –Visa, Mastercard o AmericanExpress, intestatario, numero e data di scadenza) per il pagamento. Il pagamento può anche avvenire in contanti all'atto della consegna.

Di ogni ordine, il sistema deve poter calcolare l'importo, che si ottiene nel modo usuale. In caso di importo inferiore a euro 10.00 tuttavia, viene aggiunta una tariffa di euro 3.00 per la consegna a domicilio. Inoltre, al fine di incentivare forme di pagamento elettronico, in caso di pagamento in contanti all'atto della consegna viene applicato un sovrapprezzo di euro 1.00.

La consegna non può essere richiesta per un orario distante meno di un'ora dall'ordine: in caso contrario, l'ordine deve essere rifiutato.

Gli ordini immessi seguono un ciclo di vita ben strutturato per la loro evasione: in particolare, ognuno di essi viene ricevuto da un operatore (di cui interessa il nome) che si occupa di preparare un pacco (che conterrà le specialità richieste), che sarà poi consegnato al cliente.

Alcune specialità, come ad es. le pizze o i fritti, devono essere preparate su richiesta da altri dipendenti della pizzeria (ad es. i pizzaioli), mentre altre (come le bevande o i dolci) sono subito disponibili.

Verrà data una descrizione del ciclo di vita tipico degli ordini facendo riferimento ad un esempio. Si supponga che venga ricevuto un ordine composto da: due supplì, due porzioni di olive

ascolane, una pizza boscaiola, una provola e speck, due lattine di birra e due tiramisu, oltre che due caffè rigorosamente zuccherati con dolcificanti dietetici. L'operatore assegnato all'ordine procederà con le seguenti attività: *i)* Ordina in cucina le pizze, i supplì, le olive ascolane, e i caffè; *ii)* Quando le pizze, i fritti e i caffè sono pronti, li inserisce nel pacco; *iii)* Inserisce nel pacco le birre e i tiramisu, non dimenticando le bustine di dolcificante rigorosamente dietetico. *iv)* Chiude il pacco.

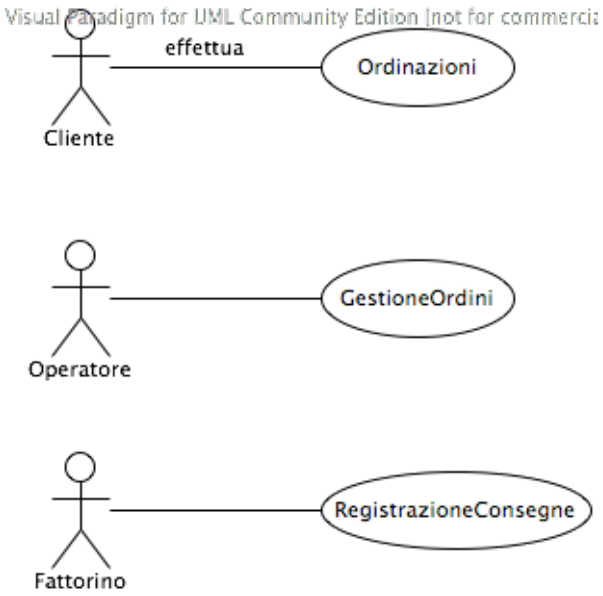
Per favorire il monitoraggio delle attività, l'operatore comunicherà al sistema ogni sua azione (ordinazioni in cucina ed inserimenti nei diversi pacchi). Questo permette di conoscere, per ogni voce di un ordine, se questa non è ancora stata considerata, è in attesa di essere preparata dalla cucina (solo per le specialità non immediatamente disponibili), oppure è stata già inserita nel pacco.

Quando un pacco diventa completo, viene assegnato dal sistema ad un fattorino per la consegna.[1] A questo punto l'ordine si definisce *evaso*. All'atto della consegna vera e propria, il fattorino, grazie ad un dispositivo palmare, comunicherà che l'ordine è stato consegnato; il sistema memorizzerà quindi l'orario di consegna al fine di consentire verifiche periodiche sulla qualità del servizio.

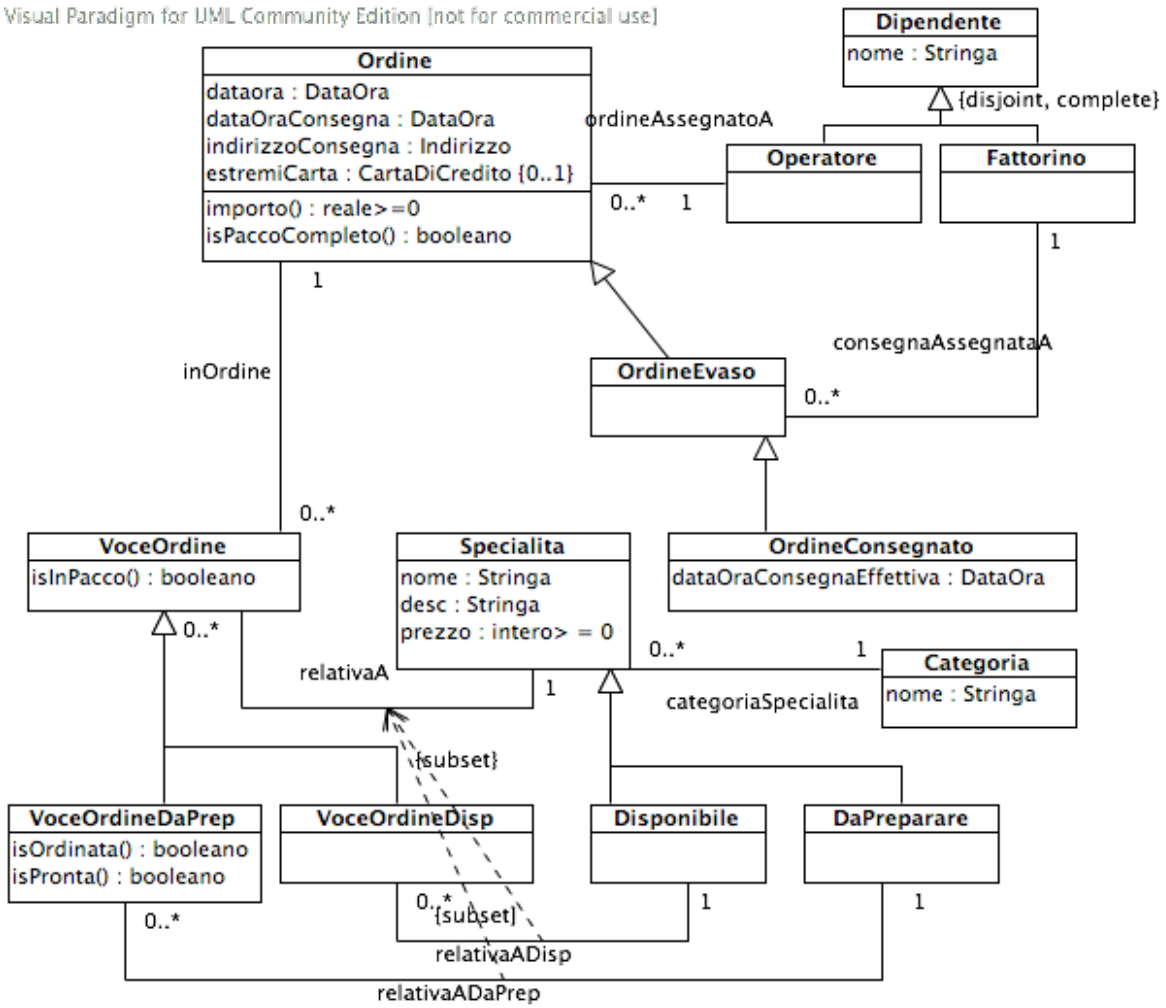
[1] Il sistema sceglierà automaticamente il fattorino da assegnare ad un pacco. Di questa funzionalità non è tuttavia richiesta la progettazione. Si assuma che esista un'opportuna operazione che la realizza; se ne scriva la segnatura, la si collochi in modo opportuno e la si utilizzi dove necessario.

1 Fase di Analisi

1.1 Diagramma degli Use Case



1.2 Diagramma delle classi UML



1.3 Specifica dei tipi di dato

```
SpecificaTipoDiDato CartaDiCredito
  attibuti
    tipo : {Visa, Mastercard, AmericanExpress}
    intestatario: Stringa
    numero: intero > 0
    dataScadenza: Data
```

```
FineSpecifica

SpecificaTipoDiDato Indirizzo
  attibuti
    via: Stringa
    nomeCitofono: Stringa
FineSpecifica
```

1.4 Specifica degli use case

```
SpecificaUseCase Ordinazioni
  iniziaOrdine(): Ordine
    pre: nessuna
    post:
        result e' un nuovo oggetto di classe Ordine, con i valori per tutti gli
        attributi indefiniti (questi saranno indicati come ``non noti alla nascita''
        nella successiva fase di progetto).

        Inoltre, result non sara' coinvolto in alcun link di qualsivoglia associazione.

        Dal diagramma degli stati della classe Ordine, deriva inoltre che result e'
        nello stato ``InDefinizione''.

  aggiungiAdOrdine(o:Ordine, s:Specialita)
     pre:  o e' nello stato ``InDefinizione''
     post:
       Se s e' di classe Disponibile, viene creato l'oggetto vod:VoceOrdineDisp
            ed i link <o,vod>:inOrdine e <vod,s>:relativaADisp;
       Altrimenti, (s e' di classe DaPreparare), viene creto v:VoceOrdineDaPrep
            ed i link <o,vop>:inOrdine e <vop,s>:relativaADaPrep.

  finalizzaOrdine(o:Ordine, consegna:DataOra, indirizzo:Indirizzo,
                                                    pagam:CartaDiCredito {0..1})
     pre:  o e' nello stato ``InDefinizione''
     post:
        - o.dataOraConsegna = consegna
        - o.indirizzoConsegna = indirizzo
        - o.estremiCarta = pagam (indefinito se pagam e' indefinito, cioe' si richiede
```

```
          pagamento in contanti)

        Viene generato l'evento ''immetti'' su ''o'', che passa allo stato ''accettato'' o
        ''rifiutato'' a seconda dell'output dell'operazine oraConsegnaOk(o)
        (cf. diagramma degli stati).

   oraConsegnaOk(o: Ordine) : booleano
      pre: o e' nello stato ''InDefinizione''
      post:  result e' true se e solo se o.consegna.differenzaOre(adesso) >= 1
FineSpecifica


SpecificaUseCase GestioneOrdini
   ordinaInCucina(vop: VoceOrdineDaPrep)
      pre: vop e' nello stato ''DaOrdinare''
      post: viene generato l'evento ''ordina'' su vop.

   mettiNelPacco(v:VoceOrdine)
      pre:  Se v e' di classe VoceOrdineDaPrep, v deve essere nello stato ''Pronta''
      post: vengono generati gli eventi ''inserisci'' su v  e
           ''specialitaNelPacco'' su vop.inOrdine.Ordine

   assegnaFattorino(o:Ordine)
     pre: o e' nello stato ''Completo''
     post:
            Sia f = scegliFattorino(o).
            - Viene generato l'evento ''assegnaFattorino'' su ''o'',
              che passa allo stato ''Evaso'' (diventando di classe ''Evaso'').
            - Viene creato il link <o, f>: consegnaAssegnataA

     scegliFattorino(o: Ordine): Fattorino
        pre: o e' nello stato ''Completo''
        post: ---non richiesto---
FineSpecifica

SpecificaUseCase RegistrazioneConsegne
  registraConsegna(o:Ordine)
    pre: o e' nello stato ''Evaso''
    post: viene generato l'evento ''consegnaACliente'' su o, che passa
         allo stato ''Consegnato''.
```

```
        Inoltre, o diventa di classe ‘‘OrdineCompletato’’, e o.dataOraConsegna = adesso.
FineSpecifica
```

1.5 Specifica delle classi e diagrammi degli stati e transizioni

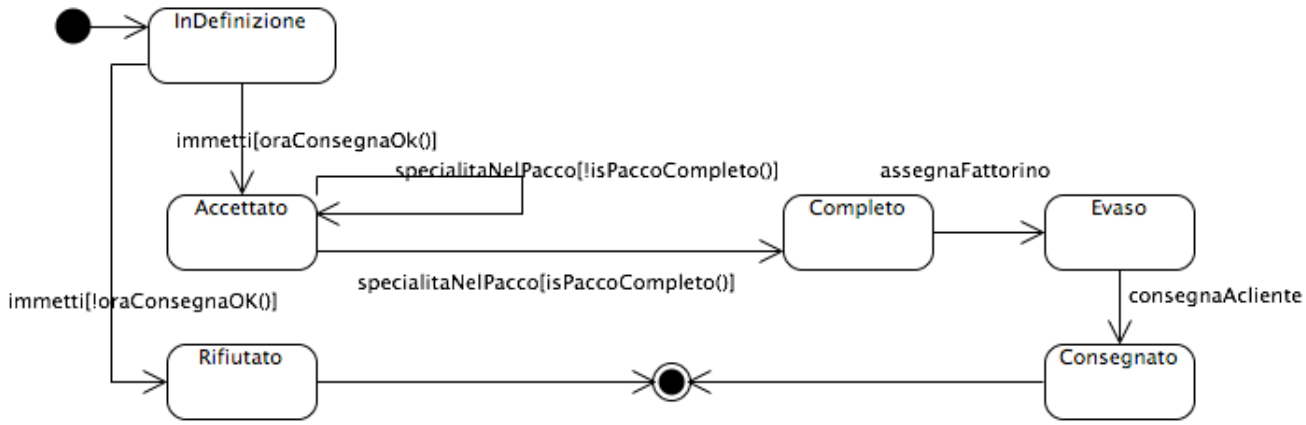
La classe Ordine

```
SpecificaClasse Ordine
  importo(): reale >= 0
    pre: nessuna
    post: Detto

      prezzoBase = Σ_{v:VoceOrdine:⟨this,v⟩∈inOrdine} v.relativaA.Specialita.prezzo

      Sia inoltre prezzoConsegna = 3.00 se prezzoBase < 10.00, 0.00 altrimenti
      Sia commContante = 1.00 se this.estremiCarta e' indefinito (pagamento in contanti),
            0.00 altrimenti.

      result e' pari a prezzoBase + prezzoConsegna + commContante.

  isPaccoCompleto(): booleano
    pre: this e' nello stato ‘‘Accettato’’
    post: result e' true se e solo se ogni v:VoceOrdine tale che <this,v>:inOrdine
         e' nello stato ‘‘InPacco’’
FineSpecifica
```

La classe VoceOrdine

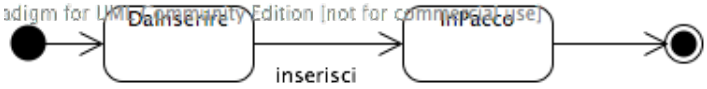
```
SpecificaClasse VoceOrdine
  isInPacco():booleano
    pre: nessuna
    post: result e' true se e solo se this e' nello stato ‘‘InPacco’’
FineSpecifica
```

Gli oggetti di questa classe evolvono secondo le regole imposte dal seguente diagramma degli stati e transizioni:



La classe VoceOrdineDisp Specifica non necessaria (nessuna operazione)

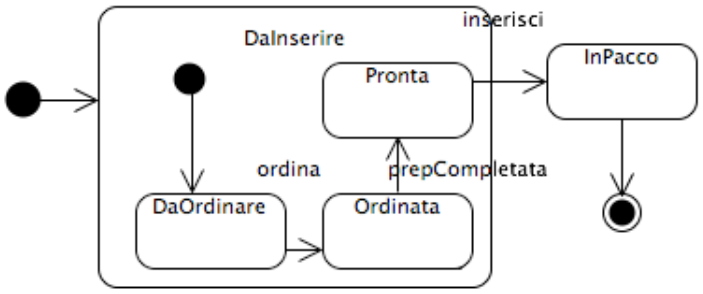
Gli oggetti di questa classe evolvono secondo le regole imposte dal seguente diagramma degli stati e transizioni:



La classe VoceOrdineDaPrep

```
SpecificaClasse VoceOrdineDaPrep
  isOrdinata(): booleano
    pre: nessuna
    post: result e’ true se e solo se this e’ nello stato ‘‘Ordinata’’

  isPronta(): booleano
    pre: nessuna
    post: result e’ true se e solo se this e’ nello stato ‘‘Pronta’’
FineSpecifica
```

Gli oggetti di questa classe evolvono secondo le regole imposte dal seguente diagramma degli stati e transizioni:


DaInserire
inserisci
InPacco
Pronta
ordina
prepCompletata
DaOrdinare
Ordinata