IntellAgent Shop:
Vendita on-line “intelligente”

System Overview

Progetto sviluppato durante il corso di:

Sistemi Intelligenti Distribuiti LS

Realizzato da:

Stefano Riffelli matr.234231

Anno Accademico 2005/2006

Abstract:

L'obiettivo di questo progetto è quello di creare un sistema ad agenti che simuli la vendita on-line "intelligente" di materiale informatico.
Più precisamente, il venditore rappresenta l'amministratore dell'intero sistema software finito ed ogni cliente che voglia fare acquisti on-line dovrà effettuare una registrazione.
Gli agenti che "girano" sotto tale sistema, monitoreranno in dettaglio tutti gli acquisti di ogni cliente ed in base a questi svolgeranno rispettivamente le seguenti funzioni:
- pubblicizzare prodotti legati agli acquisti del cliente,
- aiutare il cliente a non ordinare un prodotto sbagliato,
- proporre offerte valide solo per quel cliente.
In generale, questi agenti dovranno svolgere funzioni in base alle esigenze di ogni singolo cliente.

Requisiti:

- Mantenimento semplice, efficace e coerente delle informazioni relative ai prodotti disponibili.
- Possibilità da parte del cliente di effettuare acquisti on-line.
- Possibilità da parte del cliente di mantenere le proprie informazioni.
- Interazioni gestore/applicazione e cliente/applicazione semplici, attraverso GUI (Graphical User Interface) Web-based.
- Possibilità per il gestore di configurare i parametri degli agenti.
- Possibilità per il gestore di inserire/eliminare/modificare informazioni relative a prodotti e/o clienti.

Analisi:

Obiettivo principale del progetto “IntellAgent Shop” è la gestione informatizzata di vendita on-line mediante sistema intelligente distribuito.

In primo luogo, possiamo quindi dividere l’analisi dell’intero sistema in due sottosezioni:

- sezione distribuita: riguardante l’intero sistema di compra-vendita escluso l’acquisto del prodotto.
- sezione intelligente: riguardante il momento dell’acquisto con la rispettiva analisi degli agenti implementati.

SEZIONE DISTRIBUITA:

Il macro-obiettivo suddetto può essere scomposto attraverso una metodologia top-down in sotto-obiettivi più semplici, di seguito elencati:

- Creazione e mantenimento di un database aggiornato di prodotti resi disponibili dall'esercizio;
- Possibilità di semplice aggiornamento del database da parte del gestore;
- Interazione semplice ed efficace tra l'utente e l'esercizio attraverso interfacce grafiche;
- Possibilità per l’utente di richiedere informazioni aggiornate sui prodotti disponibili e di effettuare acquisti on-line tramite sessioni che garantiscano la protezione delle informazioni;

Uno dei pilastri fondamentali di questa applicazione è il mantenimento coerente, dettagliato, efficace e aggiornato delle informazioni relative ai prodotti disponibili all'interno dell'esercizio. Il metodo più efficace per assolvere questo compito messoci a disposizione dalle tecnologie attuali e' l'uso di un database. L'interazione con l'utente e con il gestore, che costituisce un altra colonna portante dell'applicativo, sarà gestita per mezzo di una versatile interfaccia grafica (GUI).
Possiamo pensare a questa applicazione come un progetto avente struttura biforme.

Riunisce infatti due aspetti principali e cruciali per la reale gestione di vendite on-line: il management locale, ossia la possibilità di mantenere le informazioni relative alle entità del contesto su supporto informatico, funzione che prevede tra l'altro l'inserimento, la cancellazione e la modifica di prodotti e clienti; e il management remoto che si occupa di tutti gli aspetti legati alla gestione di una vista web del negozio virtuale.

Si comincia a capire come questo applicativo si differenzi enormemente da un semplice sito web di un venditore di parti informatiche. La parola applicazione e' stata scelta di proposito per sottolineare la natura applicativa del progetto: non un semplice sito web ma una applicazione articolata che si riserva la capacità di visualizzare proprie viste attraverso pagine web.

In questo modo il gestore ha la possibilità di mantenere un database aggiornato per le proprie entità di interesse, attraverso il quale può facilmente ricercare, cancellare, aggiungere entità o compiere altre operazioni su di esse in base alle proprie necessità. Da questo punto di vista, la relase 1.0 dell'applicativo congloberà solamente alcune basilari operazioni, ma le future revisioni potranno estendere questo insieme in modo semplice ed elementare. Il gestore inoltre non ha bisogno di acquistare un'altra applicazione o di costruire un sito web nell'intento di pubblicizzare il proprio negozio o curare le relazioni con i possibili clienti. L'applicativo e' formulato in modo da svolgere anche la parte interattiva in direzione del cliente sfruttando lo stesso database. Questo, vorrei sottolinearlo, e' un ottimo risultato in quanto:

- evita al gestore nel negozio informatico di curare l'aspetto di interazione con i clienti attraverso un'applicazione ulteriore e separata,
- evita quindi l'inutile riscrittura di tutti i dati relativi alle entità di contesto,
- garantisce la consistenza delle informazioni,
- rappresenta un grande vantaggio in termini economici,
- facilita tutta una serie di operazioni.

L'ultimo punto e' particolarmente utile e vuole sottolineare come, grazie a questa architettura, sia facile compiere operazioni quali la pubblicizzazione dei prodotti,

ricerche sulle scelte dei clienti, gestione della cassa, regolazione automatica delle scorte di prodotti, etc..

Si prevede che le future applicazioni di gestione di un negozio informatico prevedano un modello analogo a quello sviluppato per questo progetto.

Internet e' una rete che domina oggi il mercato e questo avverrà ancor più in futuro. Lo stato attuale delle cose vede la Rete come centro di scambio, punto di riferimento per il commercio, il cosiddetto **e-commerce**. L'applicazione locale di gestione ormai non esiste più e non avrebbe alcuna utilità in una realtà cosi condizionata dalla condivisione in rete delle informazioni. Questo applicativo rappresenta quindi un punto di riferimento, un modello di partenza, per la modernizzazione di un negozio informatico. Un progetto di questo genere inserito nel contesto reale di un esercizio fornisce un decisivo e sostanziale contributo per due aspetti salienti:

- la gestione interna delle entità di contesto (prodotti, clienti, ...),
- il commercio.

Grazie al primo punto la gestione delle entità disponibili non costituirà più un problema. Questo non e' poco se si immagina il danno economico molto spesso causato da errori nella gestione dei propri prodotti o clienti o nella mancata ottimizzazione della stessa.

In secondo luogo sono chiari ed evidenti a tutti gli effetti benefici traducibili in un sostanziale incremento del profitto aziendale dovuti a quello che possiamo immaginare come un trasferimento della propria azienda. Se l'esercizio o l'azienda operasse a livello locale la quantità di clienti raggiunti potrebbe essere anche notevole, ma comunque limitata superiormente da vari fattori tra cui i principali senza dubbio sono la collocazione fisica dell'azienda e le distanze rispetto ai possibili utenti. Se per esempio l'azienda si trovasse in campagna, in periferia o in un luogo non facilmente raggiungibile, e' evidente che la quantità di clienti contattata sarebbe limitata ai dintorni e risulterebbe difficile pensare che raggiunga l'utenza di altre città, stati,continenti.

Dopo queste considerazioni e' possibile comprendere come l'accesso di un'azienda al

web e quindi all' e-commerce possa generare inevitabili vantaggi. Si superano i confini territoriali, la realtà non è più isolata ma estesa su larga scala a tutto il mondo. Chiunque ha accesso alla rete (con la tecnologia attuale ogni paese minimamente sviluppato) può conoscere i prodotti, diventare cliente dell'azienda, acquistare on-line, essere raggiunto dalla pubblicità. Non parliamo di teorie sconnesse dalla realta': le prime aziende che si sono affacciate sul web dando vita al nuovo mercato hanno avuto notevoli incrementi sugli indici di borsa e sulla vendita delle azioni. Lo sviluppo di un'azienda oggi dipende in sostanza dalle tecnologie che sfrutta per raggiungere i propri fini e quindi dal software, colonna portante di ogni tecnologia informatica. Un software di questo tipo applicato a un esercizio che abbia aspirazioni lungimiranti garantisce l'incremento dei clienti, quindi del fatturato, l'estensione dell'azienda e la sua crescita in una spirale evolutiva di vantaggi economici. Rappresenta quindi un investimento di cui occorre tenere conto pensando al futuro.

Dopo queste considerazioni cognitive concentriamoci sugli aspetti legati alla **struttura dell'applicativo**.

L'applicazione gestisce un negozio informatico. Le funzioni che deve assolvere l'applicazione risultano chiaramente visibili nella sezione relativa ai requisiti. Come pensare a questo software? L'utente deve interagire con esso attraverso apposite viste che ne facilitino l'uso. Quali sono le viste in gioco?

- una o più console di input per l'interazione utente/applicazione,
- una o più viste di uscita per la visualizzazione dei risultati dell'interazione.

Come modellare l'entità oggetto dell'azienda: il prodotto. Occupandosi di un negozio informatico il prodotti principali sono legati ad altri prodotti di consumo. Ad esempio e' ragionevole pensare che l'acquisto di una stampante da parte di un cliente possa comportare il successivo acquisto di una cartuccia o di un toner. L'esperienza insegna che questa considerazione non è solamente ragionevole ma è all'ordine del giorno. Occorre modellare quindi l’intera entità prodotto generica e curare la possibilità di

inserire e gestire nuove tipologie di prodotti per soddisfare le necessità variabili di un gestore o di una categoria di gestori di aziende.
Se allarghiamo la visione al contesto, scopriamo nuove entità di cui è necessario prendere coscienza. Un negozio informatico possiede un certo numero di clienti ed è plausibile che sia intenzione del gestore mantenerne le informazioni relative. Occorre quindi inserire tra le entità in gioco anche il cliente e gestirne le relative interazioni. L'analisi del prodotto nel contesto di tale negozio individua la classe che modella un prodotto generico (Prodotto).
Il modello dell'entità **Prodotto** si caratterizza per le seguenti proprietà specifiche:
- id,
- categoria,
- descrizione,
- prezzo.

Tra le entità di contesto citate si aggiunge anche il possibile cliente dell'azienda. Si modella l'entità **Cliente** attraverso le seguenti proprietà:
- fCode,
- nome,
- cognome,
- indirizzo,
- comune,
- provincia,
- stato,
- tel,
- cell,
- email,
- password,
- id.

Sembra ovvio che in futuro le entità e le relative proprietà modellate da questa analisi possano subire generici mutamenti anche in relazione ad analisi più accurate.

Considerati i limiti temporali dell'attuale progetto ci sembrano validi i modelli di cui sopra.

Passiamo ora all'analisi delle attività e delle interrelazioni tra le entità. Definiamo chiaramente quali sono le attività degli utenti dell'applicazione: il manager, il cliente, il semplice utente. Le azioni di manovra privilegiate possibili per il manager si possono definire come segue:

- inserimento/modifica/eliminazione prodotti,
- inserimento/modifica/eliminazione clienti.

Le azioni a disposizione del cliente sono usuali operazioni di acquisizione informazioni e privilegi derivanti dalla propria qualifica:

- visualizzazione/modificazione proprio profilo,
- acquisto prodotti.

Infine la terza entità interattiva, l'utente semplice, non ha privilegi particolari e la sua capacità di azione è limitata alle usuali operazioni di acquisizione informazioni:

- visualizzazione elenco prodotti,
- ricerca prodotti,
- registrazione con promozione a cliente.

Evidentemente l'utente di grado più alto e conseguentemente con la maggiore capacità di azione è il manager, segue il cliente e infine l'utente semplice. Ogni utente di livello superiore ha ovviamente la possibilità di compiere le azioni specifiche delle entità di livello subordinato, quindi ne eredita le azioni. Cosi per esempio il manager può anche visualizzare/modificare il profilo di un utente, oppure il cliente può effettuare ricerche sui prodotti o visualizzarne gli elenchi.

Possiamo anticipare già in fase di analisi che per la memorizzazione delle informazioni relative al contesto è preferibile utilizzare un database relazionale, oggi la tecnologia più diffusa e versatile in questo settore.

SEZIONE INTELLIGENTE :

In questa parte si analizzerà tutto quello che riguarda il momento dell'acquisto di un prodotto e quindi si effettuerà anche una analisi degli agenti che si vogliono realizzare.

Come appena detto, l'acquisto è quindi legato al prodotto ed al cliente; si ritiene pertanto opportuno in questa fase, considerare l'entità "Acquisto".

Si modella l'entità **Acquisto** attraverso le seguenti proprietà:

- id,
- idProdotto,
- idCliente.

Ricordando le funzioni che devono svolgere i nostri agenti, è scelta dell'analista assegnare ad ogni agente una di queste. Procedendo in questo modo avremo **3 agenti** che svolgono le seguenti funzioni:

1) **Agent Bunner:**
   guarda la categoria di prodotti più comprata da quel cliente e pubblicizza di conseguenza prodotti collegati ad essa.
2) **Agent Alert:**
   guarda il prodotto più comparto per ogni categoria di prodotti e lo confronta con quello che si sta acquistando. Se risulta essere lo stesso allora non manda in output nulla, altrimenti avvisa su cosa compra usualmente quel cliente in quella categoria di prodotti.
3) **Agent Offer:**
   guarda la spesa totale effettuata dal cliente e se il cliente ha superato una soglia (impostabile dal gestore) fa un' omaggio di un certo valore (impostabile dal gestore) sul prodotto che si sta acquistando.

Si vuole ricordare che tale sistema software è rivolto soprattutto a grandi venditori di parti informatiche che avranno quindi come clienti i "venditori diretti" di tali parti. E' evidente quindi la scelta delle azioni di Agent Bunner e Agent Offer che non avrebbe altrimenti molto senso nel caso di clienti non rivenditori (es. una comune persona che

abbia gia comprato una stampante laser, si pensa che non debba più comprarne un'altra; questo è invece altrettanto falso per un rivenditore) .

Visto il ruolo dell'Agent Offer, è chiaro ora che tra le azioni del manager, bisognerà aggiungere la seguente:

- inserimento/modifica dei parametri dell'Agent Offer.

In questa fase si citano anche altre azioni di cui il manager potrebbe averne la possibilità:

- configurazione dell'Agent Bunner (es. poter impostare i "collegamenti" tra prodotti che pubblicizzano altri prodotti oppure ad es. pubblicizzare sempre il prodotto più comprato da quel cliente)
- configurazione dell'Agent Alert (es. poter impostare la "sensibilità" con cui avvisare o meno di un eventuale acquisto errato)

In questa fase si ritiene sia giusto elencare le azioni e le possibilità che potrebbe avere questo sistema software ma che si pensa comunque di implementarne solamente alcune, per questioni temporali.

Architettura logica:

L'architettura applicativa deriva dal classico modello MVC/JSP Model 2. La parte relativa al **model** contiene il kernel dell'applicativo e si può articolare in più classi. Per comunicare con i possibili utenti, il **model** si avvale di **view**, le viste del sistema. L'interazione con l'utente è realizzata attraverso **control**, interfacce di controllo del sistema.

Come sappiamo le specifiche JSP definiscono due approcci per costruire web application utilizzando le jsp:

- JSP model 1;
- JSP model 2.

Il modello da noi utilizzato è **JSP model 2.**

Nel model 2 le richieste sono intercettate da una servlet definita ***CONTROLLER***.

Con il modello MVC permetto al servlet controller di realizzare front-end processing, autenticazione, autorizzazione, log centralizzato e internazionalizzazione.

Con l'architettura model 2 c'è una netta separazione tra business logic, presentation output e request processing.

Il pattern MVC ha tre componenti chiave:

1)Model: responsabile per la conoscenza dello stato di business;

2)View: responsabile per la vista;

3)Controller: responsabile per il controllo del flusso I/O.

Il controller in una web application realizza le seguenti operazioni:

1. Intercetta la HTTP request da parte del client (browser);
2. Interpreta ogni request in una specifica business operation da realizzare;
3. Delega la business operation ad un opportuno handler;
4. Aiuta a selezionare la view da visualizzare al client.
5. Ritorna la view al client.

Più precisamente l'applicazione viene strutturata nel seguente modo:

- le view vengono implementate mediante l'utilizzo della tecnologia **Java Server Page** che consente di disegnare gli oggetti di presentation tramite il linguaggio HTML e di inserire all'interno di essi il codice Java da eseguire lato server. Questo codice non conterrà nessuna funzione di controllo o di accesso ai dati ma solo le funzionalità di composizione e dinamicizzazione delle pagine.
- il modello viene implementato tramite l'utilizzo di **Java Beans** quando le componenti sono locali all'application server che deve gestire l'applicazione o, tramite **Enterprise Java Beans** quando le componenti sono distribuite su più application server. All'interno dei bean verranno implementate tutte le funzioni di accesso ai dati e le regole di business per il loro trattamento. All'interno dei beans non vengono inserite funzioni di controllo o di presentation.

- gli oggetti di controllo vengono implementati tramite l'utilizzo delle **servlet** che avranno il compito di rimappare le richieste dell'utente, eseguite sulle mappe generate dalle JSP, in azioni che verranno eseguite dai beans.

Il **funzionamento** è il seguente:

- l'utente interagisce con una pagina web (web form) e specifica un comando. La pagina web invia al server il comando utente (<< submit >>);
- sul server viene mandata in esecuzione la servlet di controllo che richiede ai bean (che rappresentano il modello) di eseguire le operazioni di business richieste dall'utente (<< send >>);
- la servlet di controllo seleziona la view opportuna per presentare l'output all'utente e manda in esecuzione la rispettiva Java Server Page (<< forward >>);
- la JSP recupera dal bean i dati aggiornati da visualizzare (<< call >>) quindi li elabora e genera come output una pagina HTML;
- la pagina HTML generata dalla JSP viene inviata all'utente (<< display >>).

Progetto:

In questa sede è necessario definire le scelte progettuali effettuate nella realizzazione delle entità di contesto. Le principali entità in gioco sono: prodotti, clienti e acquisti. Gli attributi di queste entità sono già stati analizzati in sede di analisi. Ciascuna entità è stata modellata attraverso una opportuna classe, le classi applicative sono quindi:

- **Prodotto,**
- **Cliente,**
- **Acquisti.**

Occorre ora specificare le proprietà di ciascuna classe che disporranno di opportuni metodi accessori (set, get*):*

Classe Prodotto:

1. id (intero),
2. categoria (stringa),
3. descrizione (stringa),
4. prezzo (virgola mobile a doppia precisione).

Classe Cliente:

1. fCode (stringa),
2. nome (stringa),
3. cognome (stringa),
4. indirizzo (stringa),
5. comune (stringa),
6. provincia (stringa),
7. stato (stringa),
8. tel (stringa),
9. cell (stringa),
10. email (stringa),
11. password (stringa),
12. id (intero).

Classe Acquisto:

1. id (intero)
2. idCliente (intero),
3. idProdotto (intero),
4. Data (stringa).

Per ragioni di coerenza nel database queste informazioni saranno rappresentate mediante gli stessi tipi di dato. Le informazioni relative ai clienti, prodotti, acquisti

devono essere mantenute su supporto informatico, funzione svolta dal database. Nel db sono presenti tre tabelle:

- prodotto,
- cliente,
- acquisto.

ciascuna delle quali contiene le informazioni relative rispettivamente ai prodotti, clienti e acquisti del negozio informatico on-line. E' possibile notare quindi che ogni classe che modella un entità di contesto *rappresenta* una tabella rispettando le seguenti convenzioni:

- il nome della classe coincide con il nome della tabella,
- i nomi e i tipi delle proprietà della classe coincidono con i nomi e i tipi degli attributi della tabella relativa,
- ogni classe suddetta estende una classe **wrapper**.

Tali classi verranno quindi chiamate **classi convenzionate**. Attraverso questo sistema è possibile delegare alla classe wrapper le operazioni di comunicazione a basso livello con il database.
L'interazione con l'utente è gestita attraverso delle viste di ingresso/uscita dei dati. Ogni vista comunica l'input ricevuto dall'utente a un apposito controller. Il controller delega la business operation relativa ad un opportuno handler e infine ritorna la vista opportuna al client. Il model business è l'entità che modella il model nel pattern MVC, ed è responsabile appunto della conoscenza dello stato di business applicativo. Ogni classe convenzionata estende la classe JSqlWrapper che mette a disposizione le funzionalità di comunicazione con il database necessarie. Le funzioni consistono nelle principali operazioni di gestione di dati attraverso database:

1. selezione di record,
2. inserimento di record,
3. aggiornamento di record,

4. eliminazione di record.

Ogni classe convenzionata come detto dispone dei metodi pubblici accessori alle proprie proprietà private.
I **casi d'uso** derivanti dalle attività messe a disposizione dall'applicazione si riducono ai seguenti:

1. ricerca di un prodotto,
2. ricerca di un cliente,
3. inserimento nuovo prodotto,
4. inserimento nuovo cliente,
5. modifica di un prodotto,
6. modifica di un cliente,
7. cancellazione di un prodotto,
8. cancellazione di un cliente,
9. visualizzazione elenco prodotti,
10. registrazione cliente,
11. acquisto prodotto,
12. modifica parametri dell'offerta ("Agent Offer")

Ogni caso d'uso vede partecipe un attore e non tutti i casi d'uso possono essere svolti da tutti gli attori possibili (per esempio un cliente o un utente non può cancellare prodotti o modificare i parametri dell' "Agent Offer"). Esaminiamo quindi ogni singolo caso d'uso per osservare il comportamento dell'applicazione.

Ricerca di un prodotto

Questo caso d'uso può avere come attori utenti semplici, clienti e manager. Attraverso la vista l'utente immette le informazioni che costituiscono i criteri di ricerca specifici di un prodotto o di più prodotti. Terminata l'immissione delle informazioni l'utente fornisce il comando di immissione. Viene quindi chiamato il controller che preleva le informazioni e modifica il model attraverso i metodi accessori delle proprietà. In questo caso la parte di model interessata ai prodotti, cioè la classe Prodotto che viene

istanziata e le cui proprietà vengono successivamente modificate tramite i metodi accessori ai relativi valori prelevati dalla vista (setId = valore, set Prezzo = valore....). Quindi il controller chiama la procedura di selezione (Prodotto.select) della classe Prodotto ereditata dal wrapper. Il wrapper si occupa di prelevare tutti i dati necessari per l'interrogazione del database dalla classe che lo estende e che partecipa al caso in studio. Infine chiama proprie procedure di basso livello per contattare il database, effettuare la richiesta dati e ricevere il risultato. Il risultato ottenuto è un set di istanze della classe convenzionata di interesse (Prodotto nel caso specifico), ognuna delle quali rappresenta un oggetto risultato che rispetta i criteri di ricerca immessi (nel caso specifico un Prodotto che rispetta i criteri di ricerca). E' possibile leggere le proprietà di ciascuna istanza risultato tramite i suoi metodi accessori. Il controller dispone la vista che accede al model per prelevare il set di istanze risultato e visualizzarlo all'utente.

Ricerca di un cliente

Come sopra a differenza che la classe in gioco è Cliente è l'unico attore di questo caso d'uso è il manager.

Inserimento nuovo prodotto

Attori possibili: manager. Il manager immette i valori per il nuovo prodotto da inserire avvalendosi della vista opportuna e conferma la sua volontà di inserire la nuova entità. Il controller preleva i dati, crea una nuova istanza della classe Prodotto del model, modificandone le proprietà relative ai valori immessi. Infine delega l'inserimento al model attraverso la chiamata del metodo *insert* del wrapper. Il wrapper preleva i dati dalla classe Prodotto del model tramite accesso indiretto alle proprietà e comunica con il database per creare il nuovo record relativo. Il controller seleziona la vista che indica il risultato (in termini di successo o fallimento) dell'operazione.

Inserimento nuovo cliente Come sopra ma la classe interessata è Cliente.

Modifica di un prodotto Attori possibili: manager. Il manager immette i valori per il prodotto da modificare avvalendosi della vista opportuna e conferma la sua volontà di

modificare la nuova entità. Il controller preleva i dati, crea una nuova istanza della classe Prodotto del model, modificandone le proprietà relative ai valori immessi tranne quella relativa alla chiave primaria. Infine delega l'aggiornamento al model attraverso la chiamata del metodo *update* del wrapper. Il wrapper preleva i dati dalla classe Prodotto del model tramite accesso indiretto alle proprietà, legge la chiave primaria del Prodotto in questione e lo identifica nel database, quindi modifica il relativo record nel database ai nuovi valori. Il controller seleziona la vista che indica il risultato (in termini di successo o fallimento) dell'operazione.

Modifica di un cliente Come sopra ma la classe interessata è Cliente.

Cancellazione di un prodotto Attori possibili: manager. Il manager immette il valore della proprietà chiave primaria per il prodotto da eliminare avvalendosi della vista opportuna e conferma la sua volontà di eliminare l'entità. Il controller preleva il dato, crea una nuova istanza della classe Prodotto del model, modificandone la proprietà relativa al valore immesso. Infine delega la cancellazione al model attraverso la chiamata del metodo *delete* del wrapper. Il wrapper preleva la chiave primaria dalla classe Prodotto del model tramite accesso indiretto alla proprietà e lo identifica nel database, quindi elimina il relativo record. Il controller seleziona la vista che indica il risultato (in termini di successo o fallimento) dell'operazione.

Cancellazione di un cliente Come sopra ma l'entità appartiene alla classe Cliente.

Visualizzazione elenco Prodotti Attori possibili: utente semplice, cliente, manager. L'utente manifesta la sua volontà di visualizzare l'elenco dei prodotti disponibili. Il controller ne prende atto e chiama la procedura di dispacciamento del model. Infine dispone la vista opportuna per la visualizzazione dell'elenco all'utente.

Registrazione cliente Ovviamente l'attore sarà l'utente semplice che dopo l'avvenuta registrazione diventerà cliente con possibilità quindi di effettuare anche acquisti.

Acquista prodotto Attori possibili: cliente e manager. L'acquisto di un prodotto è possibile solamente dopo aver effettuato il login e quindi essersi registrato ed inoltre

solamente accessibile dopo la ricerca di un prodotto o dall'elenco completo dei prodotti (una volta fatto il login).
Il cliente, avvalendosi della vista opportuna, conferma la sua volontà di acquistare il prodotto. Il controller preleva i dati (del prodotto e del cliente), crea una nuova istanza della classe Acquisto del model, modificandone le proprietà relative ai valori immessi. Infine delega l'inserimento al model attraverso la chiamata del metodo *insert* del wrapper. Il wrapper preleva i dati dalla classe Acquisto del model tramite accesso indiretto alle proprietà e comunica con il database per creare il nuovo record relativo. Il controller seleziona la vista che indica il risultato (in termini di successo o fallimento) dell'operazione.

Modifica dei parametri dell' "Agent Offer" Attori possibili: manager.

I parametri di tale Agente, come accennato in fase di analisi, sono l'inserimento di una soglia (che dovrà raggiungere il cliente prima di ottenere una offerta, come somma totale degli euro spesi) e l'inserimento dell'offerta (ossia gli euro massimi che posiamo scalare dal prodotto che sta acquistando quel cliente).

Infine come detto in fase di analisi, si potrebbero progettare anche la modifica dei parametri degli "Agent Alert" e "Agent Bunner" ma che, per ragioni temporali, non tratteremo.

Implementazione:

Date le diverse tecnologie che si hanno oggi a disposizione, è intenzione del progettista effettuare le seguenti scelte progettuali:

- Il Model è codificato in JSP.
- Il DBMS utilizzato è MySQL un dbms open-source con pieno supporto per l'integrazione a java.
- MySQL offre un connector standard java: **MySQL connector/J** per l'uso dell'interfaccia JDBC.

- Interfacciamento al dbms attraverso connector/j JDBC.

Come appena detto il Model è codificato in JSP (model 2) di conseguenza le viste sono pagine dinamiche jsp, i controller sono http servlet e il model business è codificato in classi java.
In questa fase si vogliono mostrare le parti di codice che risultano più interessanti. Ad esempio, è utile sapere il codice java all'interno della jsp "ElencoProdotti.jsp" che ci deve mostrare tutti i prodotti presenti nel Database creato con mySQL:

```
<%@ page language="java" import="BusinessObjectModel.ebeans.*"
import="java.util.Vector" %>

<%
      Prodotto result;
      for(int i = 0; i < prodotto.getBeanSet().size(); i++) {
            out.println("<tr>");
            result = (Prodotto)prodotto.getBeanSet().get(i);
            out.println("<td>"+result.getNome()+"<td>"+ result.getCategoria() +
"</td>"+"<td>"+result.getPrezzo()+"</td>");
            out.println("</tr>");
      }
%>
```

Altrettanto simile sarà il codice Java di una qualsiasi jsp che interroga il nostro Database.
Invece, nel caso in cui, si voglia accedere al Database per modificarlo (ossia accedere in scrittura), si dovrà scrivere una riga di codice rilevante che riguarda il form; più precisamente nella pagina InserisciProdotto.jsp, all'interno del body si avrà:

```
<FORM METHOD="post" ACTION="./InserisciProdotto">
```

Altrettanto simile sarà il codice Java di una qualsiasi jsp che accede in scrittura al nostro Database.
Ovviamente per accedere a tale pagina si dovrà realizzare un’area riservata in cui solamente il gestore dell'enoteca potrà accedere. Questo è possibile attraverso il concetto di sessione e realizzando una pagina di login (per l'amministratore).

Conclusioni:

Il progetto qui presentato riguarda l'incapsulamento di agenti intelligenti all'interno di **siti web dinamici**. Visti i vantaggi che offre la tecnologia **JSP** rispetto alle altre nella realizzazione di **web service** e data la sua notevole diffusione anche in ambiti extra accademici, si è scelto di utilizzarla per lo sviluppo della maggior parte delle pagine del sito web dinamico di test, garantendo di fatto all'ambiente notevoli elasticità e dinamicità.

Per quanto riguarda, invece, la scelta della **base di dati**, si è optato per **mySQL**, un motore free e liberamente utilizzabile, per via dell'estrema velocità di risposta ed esecuzione delle query, la notevole diffusione e la grande knowledge base già esistente sul web relativa all'utilizzo e l'esecuzione delle più svariate attività.

A riguardo infine del linguaggio per l'implementazione degli **agenti**, la scelta è ricaduta su **Java**, dato il pieno supporto alla realizzazione di web service che il linguaggio offre attraverso la tecnologia JSP, permettendo di fatto di incapsulare agenti intelligenti all'interno di semplici classi.